Ph.D. DISSERTATION

Attentional Action-Driven Deep Networks for Visual Object Tracking

주시 행동 기반 물체 추적 심층 신경망

BY

Sangdoo Yun

AUGUST 2017

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Attentional Action-Driven Deep Networks for Visual Object Tracking

주시 행동 기반 물체 추적 심층 신경망

지도교수 최 진 영

이 논문을 공학박사 학위논문으로 제출함

2017 년 8 월

서울대학교 대학원

전기 컴퓨터 공학부

윤 상 두

윤 상 두의 공학박사 학위논문을 인준함

2017 년 8 월

위 원 장	조 남 익	
부위원장	최 진 영	
위 원	오 성 회	
위 원	곽 노 준	
위 원	이 민 식	

Abstract

This dissertation proposes a novel visual tracking method which is controlled by sequential actions learned by deep reinforcement learning. In the recent trackers using deep networks, tracking-by-detection scheme is adopted to select the target position with the highest matching score. The tracking-by-detection scheme achieves a good performance in a simple manner but is inefficient in exploring candidates. We propose an efficient action-driven deep tracker which is controlled by sequential actions trained by deep reinforcement learning. In contrast to the existing trackers using deep networks, the proposed tracker is designed to achieve a light computation as well as satisfactory tracking accuracy in both location and scale. The deep network to control actions is pre-trained using various training sequences and fine-tuned during tracking for online adaptation to target and background changes. The pre-training is done by utilizing deep reinforcement learning as well as supervised learning. The use of reinforcement learning enables even partially labeled data to be successfully utilized for semi-supervised learning. In addition, this dissertation tackles a tracking problem of an object interacting with other objects in a complex scene such as basketball game scenes containing various interactions among players and painting motions. For this purpose, we design a multi-agent architecture diverse interaction movements among neighboring objects near the target object. The multi-agent architecture is designed so that the main tracker could determine a proper action by utilizing the states of neighboring trackers. Through extensive evaluation, the proposed tracker is validated to achieve a competitive performance that is much faster than state-of-the-art deep network-based trackers.

Keywords: visual tracking, convolutional neural network, deep reinforcement learning

Student Number: 2013-30245

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The dissertation focuses on the *visual object tracking* which solves the problem of localizing a moving object from consecutive video frames. Visual tracking is one of the fundamental problems in the computer vision field and can be used for various applications such as visual surveillance, sports analytics, and autonomous driving as shown in Figure 1.1. It is difficult to localize the target object in tracking videos because of several tracking obstacles such as motion blur, occlusion, illumination change, and background clutter as shown in Figure 1.2. Conventional tracking methods [1–9] try to capture the target objects using low-level hand-crafted features. Although they achieve computational efficiency and comparable tracking performance, they are still limited in solving the above-mentioned tracking obstacles because of their insufficient feature representation.

Recently, tracking methods [11–13] using convolutional neural networks (CNNs) have been proposed for robust tracking and vastly improved tracking performance with

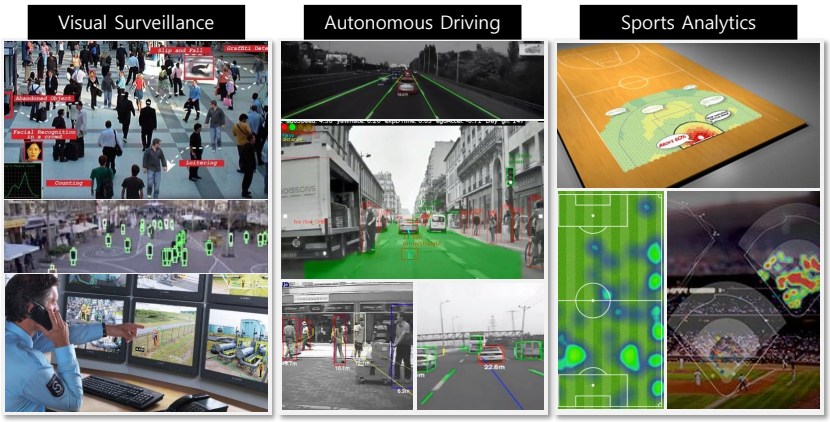


Figure 1.1: Applications of visual object tracking. Visual object tracking can be used for divergent applications such as visual surveillance, sports analytics, and autonomous driving.

the help of rich feature representation. Several algorithms [11, 12] utilize pre-trained CNNs on a large-scale classification dataset such as ImageNet [14]. However, due to the gap between classification and tracking problem, the pre-trained CNN is not sufficient to solve the difficult tracking issues. Nam *et al.* [13] proposed a tracking-by-detection algorithm with CNNs trained with tracking video datasets such as [10, 15] and achieved outperforming performance compared to the existing trackers. However, since this approach usually focuses on improving the ability to distinguish the target and background using the appearance model, it has inefficient search strategy that explores the region of interest and selects the best bounding box covering the target by matching with the tracking model. To overcome this search problem of the tracking-by-detection methods, we introduce an action-driven tracking mechanism that actively pursues the target object considering the context of an image.

In addition, there is a critical problem in constructing the training data for Deep CNN-based tracker. Deep CNN-based trackers require a large amount of training data

Figure 1.2: Difficulties of visual object tracking problem. It is difficult to localize a target in consecutive frames by deformation, occlusion, illumination change, motion blur, etc. The examples in this figure are sampled from [10].

to learn the convolutional feature representation. Even though there are plenty of video sequences, it is extremely expensive to annotate the target position in every frame for the construction of training data. If we can train the deep CNN-based tracker using a partially labeled video sequence, variety of videos can be utilized with less effort for the training. However, the existing deep CNN-based trackers [11–13] adopt the supervised learning scheme and so have difficulties in utilizing the partially-labeled video sequences. the partially-labeled video sequences. In this dissertation, in contrast to the existing deep CNN-based trackers using supervised learning, we try to develop a reinforcement learning scheme to utilize the partially labeled video sequences effectively for training our action-driven deep tracker.

While the proposed action-driven tracking method achieved satisfactory performance, it is still a very challenging problem to track the target object in the complex situation such as crowded scenes or sports scenes. In the complex scene, it is difficult to understand and track the movement of the target in a local view because the ob-

jects interact with each other and have complicated moving patterns. To consider the interactions among objects in a complex scene, global context-aware tracking methods have been proposed for the purpose of single object tracking [16, 17] and multiple object tracking [18, 19]. In this dissertation, to solve the *interaction* problem, we additionally extend the action-decision network to consider the interaction motions among neighboring objects.

1.2 Related Work

1.2.1 Visual Object Tracking

As surveyed in [20, 21], various trackers have shown their performance and effectiveness on various tracking benchmarks [10, 15, 22]. The approach based on Tracking-by-Detection [1–4] aims to build a discriminative classifier that distinguishes the target from the surrounding background. Typically these methods capture the target position by detecting most matching position using the classifier. Online boosting methods [1,2] were proposed to update the discriminative model in online manner. Multiple instance learning (MIL) [3] and tracking-learning-detection (TLD) [4] methods were proposed to update tracking model robust to the noise.

Tracking methods based on correlation filter [5–9] have attracted attention due to their computational efficiency and competitive performance. This approach learns the correlation filter in a Fourier domain with low computational load. Bolme *et al.* [5] proposed a minimum output sum of squared error (MOSSE) filter and Henriques *et al.* [7] proposed kernelized correlation filters (KCF) with multi-channel features. Hong *et al.* [8] proposed the combined system employing short-term correlation tracker and long-term memory stores. Choi *et al.* [9, 23] proposed the integrated system of the various type of correlation filters with attentional weighted map. To overcome the insufficient representation of the hand-crafted features, deep convolutional features are

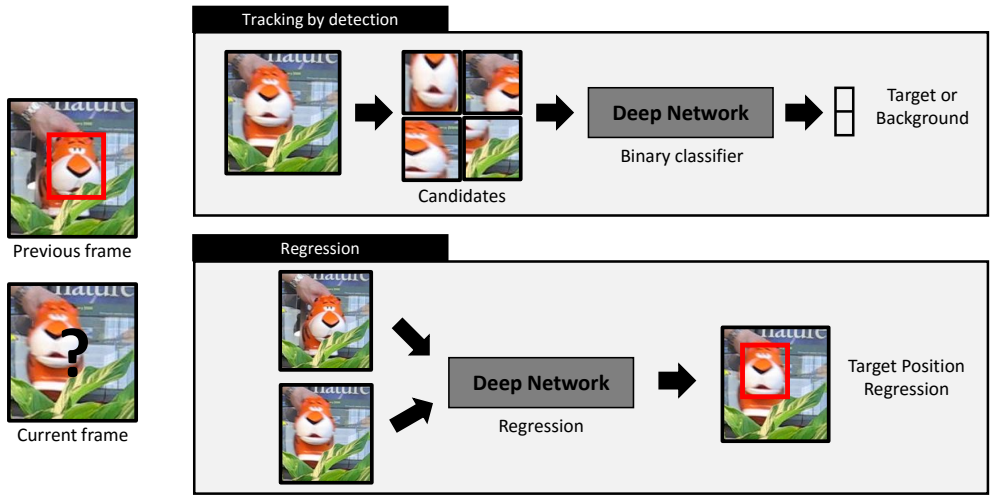


Figure 1.3: Conventional CNN-based tracking methods. Two main approaches to utilize deep neural network for visual tracking are *tracking-by-detection* and *regression* method. The detailed description of the deep tracking models are presented in Section 1.2.1.

utilized in the correlation filters [24, 25] which have achieved the state-of-the-art performance. However, since they need to train various scale-wise filters to deal with the scale change and compute the deep features, they are much slower than the traditional correlation filter based methods.

Recently, CNN-based methods [11–13, 26–31] have been proposed to learn the tracking models. Early attempts [26–28] were suffering from the data deficiency problem for training their networks. To solve the insufficient data problem, transferring methods [11, 12] were proposed by utilizing the pre-trained CNNs on a large-scale classification dataset such as ImageNet [14]. However, these methods still have a limitation due to the gap between the object classification and tracking domain.

Recently proposed methods [13, 30, 31] overcome the gap by training their network with a large amount of tracking video datasets [10, 15, 21]. Figure 1.3 briefly describes the recently proposed tracking methods: *tracking-by-detection* and *regression* approaches. In the tracking-by-detection method, the deep neural network is used to classify whether the candidate patch is located to the target or background. In the regression approach, the deep network takes the previous frame (or the initial frame) and the current frame as its input and the target position is regressed by the network.

Held *et al.* [31] proposed the tracking algorithm which captures the target's location with deep regression networks. However, this method has difficulties in tracking the target when the target is moving too quickly or occlusion is happened since it has no online updating procedure. Tao *et al.* [30] and Nam *et al.* [13] proposed Tracking-by-Detection approach that distinguishes the target and the surrounding background using the trained CNNs and successfully achieved the state-of-the-art performance. However, these methods [13, 30] need computationally inefficient search algorithms, such as sliding window or candidate sampling.

1.2.2 Action-Driven Approach

Action-driven mechanism is actively applied to various computer vision applications [32–37]. Mnih *et al.* [32] proposed an attentional method to localize MNIST digit in an image using recurrent neural network. Gonzalez-Garcia *et al.* [33] proposed an active strategy to choose the search window for object detection using the image context. Yoo *et al.* [34] proposed a deep CNN framework (AttentionNet) to capture the object by sequential actions with top-down attention. AttentionNet has achieved satisfactory performance on object detection benchmark, PASCAL VOC 2007 and 2012 [38], by sequentially refining the bounding boxes. Mather *et al.* [36] proposed a sequential search strategy to detect visual objects in images, where the detection model is trained by reinforcement learning. Caicedo *et al.* [35] proposed a deep reinforcement

learning framework to select the proper action to capture the object in an image. Xiang *et al.* [37] addressed the multiple object tracking problem as decision-making problem. However, this method learns the decision process to switch the status of the tracker, such as tracking success or tracking failure, which is different from our scheme learning the tracking actions depending on the context of the image. In visual tracking problem, action-driven tracking mechanism considering image context using deep convolutional feature has not been proposed yet to our best knowledge.

1.2.3 Deep Reinforcement Learning

The goal of reinforcement learning (RL) is to learn a policy that decides sequential actions by maximizing the cumulative future rewards [39]. Recent trends [40–43] in RL field is to combine the deep neural networks with RL algorithms by representing RL models such as value function or policy. By resorting of the deep features, many difficult problems such as playing Atari games [40] or Go [43] can be successfully solved in semi-supervised setting.

There are two popular approaches in deep RL algorithms: Deep Q Networks (DQN) and policy gradient. DQN is a form of Q-learning with function approximation using deep neural networks. The goal of DQN is to learn a state-action value function (Q), which is given by the deep networks, by minimizing temporal-difference errors [40]. Based on the DQN algorithm, various network architectures such as Double DQN [41] and DDQN [44] were proposed to improve performance and keep stability. Gu *et al.* [45] proposed a continuous deep Q learning with model-based acceleration.

Policy gradient methods directly learn the policy by optimizing the deep policy networks with respect to the expected future reward using gradient descent. Williams *et al.* [46] proposed REINFORCE algorithm simply using the immediate reward to estimate the value of the policy. Silver *et al.* [42] proposed a deterministic algorithm to improve the performance and effectiveness of the policy gradient in high-dimensional ac-

tion space. Lillicrap *et al.* proposed Deep Deterministic Policy Gradient (DDPG) [47] to apply policy gradient in continuous action space. In the work of Silver *et al.* [43], it is shown that pre-training the policy networks with supervised learning before employing policy gradient can improve the performance.

The successful deep RL methodology is applied to other computer vision application, such as object localization [33, 36] or action recognition [48], but has not been attempted in visual tracking. In this dissertation, we train the proposed tracking model with combined learning method using supervised learning and reinforcement learning. In supervised learning, the appearance characteristics of the target objects are trained. Additionally, action dynamics of the tracking target are trained with reinforcement learning using policy gradient method.

1.2.4 Multi-agent Reinforcement Learning

The multi-agent reinforcement learning has been constantly studied area in machine learning. Some traditional methods solve the multi-agent RL problem by defining the communication protocol between the agents [49–51] rather than learning the protocol. Kasai *et al.* proposed inter-agent communication learning method using tabular Q-learning [52]. To take the advantage of the deep representation, multi-agent RL methods using deep neural networks have been proposed [53–58]. According to the purpose of the reinforcement learning, the deep multi-agent RL methods can be categorized: sports player analysis using inverse reinforcement learning [57], multiple object detection [58], playing computer game (StarCraft) [56]. To train the multi-agent RL, He *et al.* [54] proposed an opponent modeling and Foerster *et al.* [53] proposed a learning algorithm of inter-agent communication by designing differentiable deep network architecture.

In this dissertation, we define an agent by the proposed action-decision network and develop a multi-agent system composed of multiple action-decision networks with

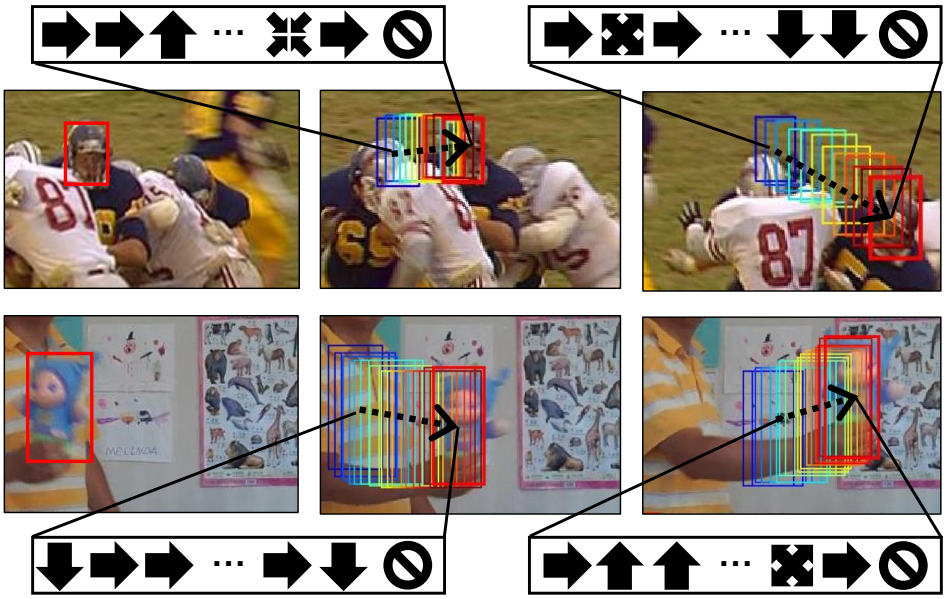


Figure 1.4: The concept of the proposed action-driven visual tracking method. **Left:** the initial frame and the initial position of the target. **Middle and Right:** localizing the target by selecting sequential actions.

communication to each other. The multi-agent system is inspired by the differentiable multi-agent model [53]. The communication module, which receives the information of the target agent and the other agents, is defined as a differentiable layer so that learning is possible.

1.3 Contents of Research

In this dissertation, the proposed action-driven deep tracker pursues the change of target by repetitive actions controlled by the action-decision network (ADNet) with deep CNN architecture. We cast the visual tracking problem as selecting the sequential actions and suggest a training method using reinforcement learning. The basic concept

of the proposed visual tracking is depicted in Figure 1.4. In Figure 1.4, the first column shows the initial location of the target, and the second and third columns show the iterative action flow to find the target bounding box in each frame. The sequential actions selected by the proposed method control the tracker to iteratively move the initial bounding box (blue) to the target bounding box (red) in each frame. The ADNet is designed to generate sequential actions to find the location and the size of the target object in a new frame. The ADNet learns the policy that selects the optimal actions to track the target from the state of its current position. In ADNet, the policy network is designed with a convolutional neural network [59], in which the input is an image patch cropped at the position of the previous state and the output is the probability distribution of proper actions including translation and scale changes. This action-selecting process has fewer searching steps than sliding window or candidate sampling approaches [13, 30]. In addition, since our method can precisely localize the target by selecting actions, post-processing such as bounding box regression [13] is not necessary.

We propose a learning algorithm with deep reinforcement learning to train the ADNet. The whole training framework is composed of supervised learning (SL) stage and reinforcement learning (RL) stage. In the SL stage, we train our network to select actions to track the position of the target using samples extracted from training videos. In this step, the network learns to track general objects without sequential information. In the RL stage, the trained network in the SL stage is used as an initial network. We obtain training samples for reinforcement learning by performing tracking simulation on training sequences. The network is trained with deep reinforcement learning based on policy gradient [46], using the rewards obtained during the tracking simulation. Even in the case where training frames are partially labeled (semi-supervised case), the proposed framework successfully learns the unlabeled frames by assigning the rewards according to the results of tracking simulation.

In addition, to solve the *interaction* problem, we propose a multi-agent extension of the action-decision network that jointly determines actions for multi-agents interacting with each other. Pursuing joint action of multiple agents considering their interactions can help tracking problem in a complex scene. A simple extension of the single-agent method to a multi-agent version can not model the interaction between agents. Our approach should maintain reinforcement learning (RL) framework when extending to multi-agent system to take advantage of the reinforcement learning framework. The multi-agent reinforcement learning framework selects joint actions from the current states of the multiple agents. The two main problems of multi-agent RL are to make communication between different agents and to jointly learn an optimal policy for all agents. Learning of joint communication among agents is a difficult task since the action and state space increase exponentially according to the number of agents. To handle the dimensional issue, the proposed framework adopts inter-agent communication manner [53]. Each agent of our method individually determines the action by its policy considering the *messages* from the other agents. The proposed multi-agent action-decision network has two input signals: the target state and the messages from the other agents.

The main contributions of the thesis are summarized as follows. The action-driven deep tracker is proposed for the first time to dynamically track the target object by pursuing-actions instead of tracking by detection scheme. We cast the visual tracking problem as Markov decision process and design deep network architecture to realize the decision process. Deep reinforcement learning algorithm is developed to train the action-decision network with partially labeled data in semi-supervised setting. The proposed deep tracker can control the trade-off between tracking performance and computational complexity by simply changing the meta parameter in tracking. The proposed tracker achieves a state-of-the-art performance with much efficient searching complexity than the existing deep network-based trackers employing a tracking-by-

detection strategy. Also, the fast version of the proposed method operates in real-time on GPUs, outperforming the state-of-the-art real-time trackers. In the multi-agent version of action-decision network, we propose a novel multi-agent framework to track the multiple target objects with action-driven manner. We also formulate multi-agent reinforcement learning problem and provide training algorithm of multi-agent reinforcement learning. The proposed multi-agent action-driven tracker achieves outperforming performance compared with the single-agent action-decision network and the state-of-the-art trackers.

1.4 Thesis Organization

This dissertation is organized as follows. In Chapter 2, we discuss basic reinforcement learning theory as preliminaries. We first describe the definition of Markov decision process and reinforcement learning problem. Then we describe optimization method to obtain the optimal policy of the reinforcement learning system. Chapter 3 presents the proposed action-driven visual object tracking scheme in the single-agent case. In this chapter, the detailed description of the proposed method and training scheme utilizing deep reinforcement learning are presented. Chapter 4 describes a multi-agent extension of the proposed action-driven tracking method. We present the detailed explanation of the proposed multi-agent system and training scheme with deep reinforcement learning. In Chapter 5, we evaluate the proposed tracker with extensive experiments and present analysis and discussion of the experimental results. In Chapter 5.1, we demonstrated the single-agent action-decision network on visual tracking benchmarks. In Chapter 5.2, we tested the multi-agent action-decision network on complex sports dataset. Finally, we conclude this dissertation by summarizing the contribution and mentioning the direction of the future research in Section 6.

Chapter 2

Preliminaries

In this chapter, we introduce the main theories used in this dissertation. In Chapter 2.1, we describe the theory of reinforcement learning with Markov decision process. In Chapter 2.2, we describe and derive about the policy gradient optimization method.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a field in machine learning to solve the sequential decision making problem. In Reinforcement learning, an agent interacts with the environment and receives observation and reward according to the chosen action at each time step as shown in Figure 2.1. In general, reinforcement learning aims to maximize the agent's excepted total reward through the sequential process. We first define the basic Markov Decision Process which constitutes reinforcement learning in Chapter 2.1.1, then define the reinforcement learning objectives via Markov Decision Process in Chapter 2.1.2. Chapter 2.2 describes policy optimization to solve the reinforcement learning problem, especially policy gradient approach which is used by the

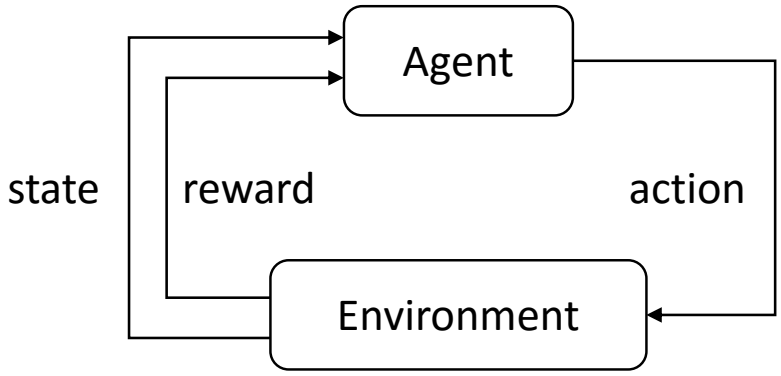


Figure 2.1: Reinforcement learning scenario.

proposed method in this thesis.

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) provides a mathematical framework to model the decision making of an agent interacting with a stochastic environment. MDPs are usually solved via *Dynamic Programming* or *Reinforcement Learning* and used in a wide area of fields, including robotics, automated control, economics, and manufacturing. A formal MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P(\cdot, \cdot))$ with the components:

- $\mathcal{S}$ (**state space**): a set of states of the environment.
- $\mathcal{A}$ (**action space**): a set of actions, where an agent selects action $a \in \mathcal{A}$ at each time step.
- $P(r, s'|s, a)$ (**transition probability distribution**): For given state $s \in \mathcal{S}$ and action a, the transition probability P specifies the probability of the reward r and transition to state s'.

The transition probability distribution satisfies *the Markov property*. That is, the response of the environment at time step $t+1$ (reward and state transition) depends only on the state and the action at time step t.

$$P(r_t, s_{t+1}|s_t, a_t) = P(r_t, s_{t+1}|s_0, a_0, ..., s_{t-1}, a_{t-1}, s_t, a_t) \tag{2.1}$$

Various different definitions are used throughout the literature. For example, the reward function can be defined as $R(s)$, $R(s, a)$, or $R(s, a, s')$ corresponding to reinforcement learning method. In the thesis, we use $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ since we use the reward by the action from the given state, and also we use $P(s'|s, a)$ as transition probability distribution.

Let π denote stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ which is represented as state conditional distribution. The goal of MDP is to find optimal **policy** $\pi(a|s)$ which makes decision of actions from given states.

2.1.2 Reinforcement Learning Problem

Generally, there are various approaches to reinforcement learning problem depending on what the objectives are: policy, value function, or dynamics models. In this thesis, we focused on policy optimization problem using *episodic* reinforcement learning setting. An episode is defined as a sequence composed of finite number of states, actions, and rewards. Each episode is started from the initial state of the environment s_0 from distribution $\mu(s_0)$. At each time step $t = 0, 1, 2, ...$, the agent selects an action a_t sampled from distribution $\pi(a_t|s_t)$. The policy π is a conditional probability distribution that the agent uses to sample its action. After deciding the action a_t, the environment generates the next state s_{t+1} and the reward r_t according to the transition probability $P(s_{t+1}|s_t, a_t)$. The episode ends when the agent reaches the terminal state s_T. This

decision process can be described as followings,

$$
\begin{aligned}
s_0 &\sim \mu(s_0), \\
a_0 &\sim \pi(a_0|s_0), \\
r_0 &\sim R(s_0, a_0) \\
s_1 &\sim P(s_1|s_0, a_0), \\
a_1 &\sim \pi(a_1|s_1), \\
r_1 &\sim R(s_1, a_1) \\
&... \\
s_{T-1} &\sim P(s_{T-1}|s_{T-2}, a_{T-2}), \\
a_{T-1} &\sim \pi(a_{T-1}|s_{T-1}), \\
r_{T-1} &\sim R(s_{T-1}, a_{T-1}) \\
s_T &\sim P(s_T|s_{T-1}, a_{T-1}).
\end{aligned}
\tag{2.2}
$$

The goal of reinforcement learning is to find a policy π that maximizes the expected total reward per episode.

$$
\begin{aligned}
&\underset{\pi}{\text{maximize}} \;\; \mathbb{E}_\tau[R|\pi] \\
&\text{where} \;\; R = r_0 + r_1 + ... + r_T
\end{aligned}
\tag{2.3}
$$

The expected total reward is computed over trajectory τ which is composed of sequences of states, actions, and rewards,

$$
\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T). \tag{2.4}
$$

The trajectory is generated by choosing sequential actions using policy π.

2.2 Policy Gradient

In this thesis, we use *parameterized stochastic policy* which is written as $\pi_\theta(a|s)$ where the parameter $\theta \in \mathbb{R}^d$ determines the policy of an agent. There are many ways to

approximate policy, in this dissertation, we use a neural network based approximation method to resort of the representation performance of deep learning. In the neural network, $\theta \leftrightarrows$ flattened weights and biases of the network. The architecture of neural network representing policy has various choices according to the MDP setting. We define the action in the discrete space and the input of neural network is defined as state s and the output of network as probability distribution $\pi(a|s)$. To make the output of neural network be the probability distribution, the final layer of the network is *softmax* layer.

The optimization formulation of the episodic reinforcement learning is defined as the following,

$$\underset{\pi}{\text{maximize}} \;\; \mathbb{E}[R|\pi], \tag{2.5}$$

where R denotes the total reward of an episode. Using the parameterized model π_θ for the policy, Eq.(2.5) becomes an optimization problem with respect to $\theta \in \mathbb{R}^d$.

$$\underset{\theta}{\text{maximize}} \;\; \mathbb{E}[R|\pi_\theta] \tag{2.6}$$

Here, we will describe the *policy gradient* method to solve the policy optimization for reinforcement learning. Policy gradient is a learning method that iteratively estimates the gradient of policy with respect to the parameter θ and finds the optimal policy. To derive policy gradient, we use *score function gradient estimator*, estimating gradients of the expectation. x is random variable with probability density function $p(x|\theta)$, f is a scalar-valued function, then ∇_θ is computed as followings,

$$\begin{aligned}
\nabla_\theta \mathbb{E}_x[f(x)] =& \nabla_\theta \int f(x)p(x|\theta)dx = \int f(x)\nabla_\theta p(x|\theta)dx \\
=& \int f(x)p(x|\theta)\nabla_\theta \log p(x|\theta)dx \\
=& \mathbb{E}_x[f(x)\nabla_\theta \log p(x|\theta)].
\end{aligned} \tag{2.7}$$

To obtain the estimation of the gradients, N samples are generated from $x \sim p(x|\theta)$ and compute Eq.(2.7) by averaging over N samples. Then the gradient estimate $\hat{g}$ using

N samples is

$$\hat{g} = \frac{1}{N} \sum_{n=1}^{N} f(x_i) \nabla_\theta \log p(x_i|\theta). \tag{2.8}$$

With this gradient estimation, policy gradient is derived in *stochastic* policy setting. That is, at each state s, our policy gives us a probability distribution over actions $\pi(a|s)$. In our case, the policy has a parameter θ and the policy is written by $\pi_\theta(a|s)$ or $\pi(a|s, \theta)$.

We define trajectory τ as sequences of states and actions, $\tau = (s_0, a_0, ..., s_T)$, $p(\tau|\theta)$ is the probability of the entire trajectory sequences τ under the parameters of the policy θ. We also denote $R(\tau)$ as the total reward of the trajectory τ. Using Eq.(2.7), the derivation of the gradient estimation with the trajectory τ and total reward $R(\tau)$ is

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}[R(\tau) \nabla_\theta \log p(\tau|\theta)]. \tag{2.9}$$

Using the chain rule and Markov property, $\log p(\tau|\theta)$ is expanded as following,

$$\begin{aligned} p(\tau|\theta) = \mu(s_0)\pi(a_0|s_0, \theta)P(s_1|s_0, a_0)\pi(a_1|s_1|\theta) \\ ... \ \pi(a_{T-1}|s_{T-1}, \theta)P(s_T|s_{T-1}, \theta), \end{aligned} \tag{2.10}$$

where μ is the initial state distribution. We take the logarithm to Eq.(2.10), and differentiate with respect to θ to drop out the terms $P(s_t|a_{t-1}, a_{t-1})$ and $\mu(s_0)$. Then we obtain the policy gradient

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) R(\tau) \right]. \tag{2.11}$$

From the above equations, we do not need the system dynamics to obtain the policy gradient estimation. In order to learn the optimal policy, we collect a trajectory and increase the log-probability multiplied by the reward of the trajectory. For example, when the reward $R(\tau)$ is high, we update the parameter θ by the gradient direction to increase $p(\tau|\theta)$.

Here, we describe REINFORCE [46] algorithm, a episodic policy gradient method, which is used in this dissertation to learn optimal policy. As shown in Algorithm 1, REINFORCE algorithm uses Monte-Carlo method to learn policy. In each iteration, a trajectory is generated by the policy π and policy parameter θ is updated using policy gradient theory. That is, after collecting an episode, the algorithm compute return v_t according to each state s_t using received rewards during the trajectory to update the policy parameter θ. The return v_t can be defined as intermediate reward r_t, expected long-term reward $\sum_{t'=t}^{T} r_{t'}$, or discounted expected long-term reward $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ where $\gamma < 1$.

In general, the policy gradient method may suffer the high variance problem of gradient estimation. In many policy gradient algorithm, *baseline* is used to decrease the variance. To adopt baseline to REINFORCE algorithm, we replace the return value v_t into the subtracted value with $b(s_t)$ as the following,

$$\theta \leftarrow \theta + \alpha(\mathrm{v}_t - b(s_t))\nabla_\theta \log \pi(a_t|s_t, \theta). \tag{2.12}$$

Adding the baseline term $b(s_t)$ does not affect on the expectation formulation as the followings,

$$\begin{aligned}
&\mathbb{E}\left[\nabla_\theta \log \pi(a_t|s_t,\theta) b(s_t)\right] \\
=&\mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi(a_t|s_t,\theta) b(s_t)\right]\right] \\
=&\mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi(a_t|s_t,\theta)\right]\right] \\
=&\mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_t|s_t,\theta)\right]\right] \qquad \text{(removed irrelevant terms)} \\
=&\mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\cdot 0\right] \\
&\text{(because, } \mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_t|s_t,\theta)\right] = \nabla_\theta \mathbb{E}_{a_t}[1] = 0)
\end{aligned} \tag{2.13}$$

Selecting baseline $b(s)$ is divergent as summarized in [60]. In practice, generally

the state-value function $V^{\pi}(s)$ is used to approximate baseline.

$$b(s) \approx V^{\pi}(s) \tag{2.14}$$

Algorithm 1 Monte-Carlo Policy Gradient (REINFORCE).

Initialize policy parameters θ

repeat

  Generate a trajectory $\tau : (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T) \sim \pi(\cdot|\theta)$

  for each time step t **do**

    Compute *return* v_t

    Update the policy parameter $\theta \leftarrow \theta + \alpha \mathrm{v}_t \nabla_\theta \log \pi(a_t|s_t, \theta)$

  end for

until θ converges

Algorithm 2 REINFORCE algorithm with baseline.

Initialize policy parameters θ

repeat

  Generate a trajectory $\tau : (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T) \sim \pi(\cdot|\theta)$

  for each time step t **do**

    Compute *return* v_t

    Update the policy parameter $\theta \leftarrow \theta + \alpha(\mathrm{v}_t - b(s_t)) \nabla_\theta \log \pi(a_t|s_t, \theta)$

  end for

until θ converges

Chapter 3

Action-Driven Visual Tracking

3.1 Overview

In this chapter, we describe the *action-driven* approach to track the target object in video frames. Visual tracking solves the problem of finding the position of the target in a new frame from the current position. The proposed tracker dynamically pursues the target by sequential actions controlled by the Action-Decision networks (ADNets) shown in Figure 3.1 (Details are in Section 3.2).

The proposed networks predict the action to chase the target from the position of the current tracker. The tracker is moved by the predicted action from current state, and then the next action is predicted from the moved position. By repeating this process over the test sequence, we solve the object tracking problem. The ADNet is pre-trained by supervised learning (Section 3.4.1) as well as reinforcement learning (Section 3.4.2). During actual tracking, online adaptation (Section 3.5) is conducted.

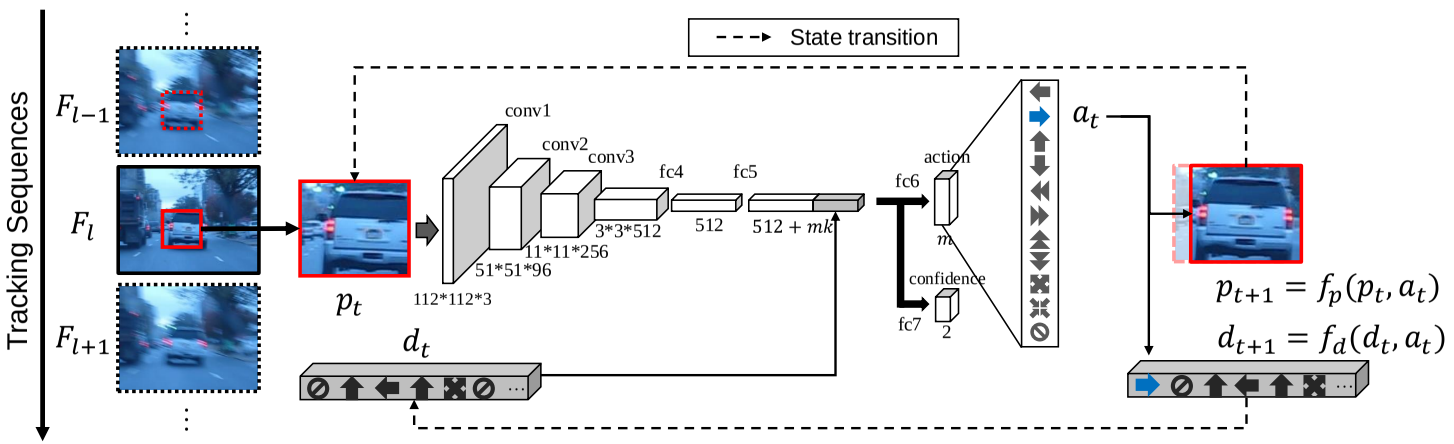


Figure 3.1: Architecture of the proposed network. The dashed lines indicate the state transition. In this example, the 'move right' action is selected to capture the target object. This action-decision process is repeated until finalize the location of the target in each frame.

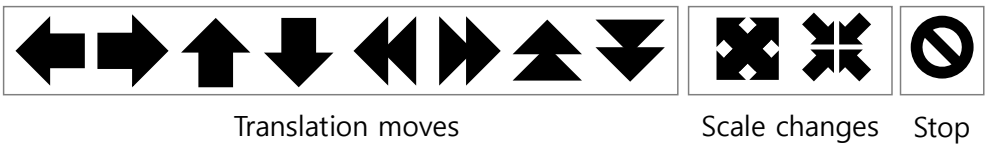


Figure 3.2: The defined actions in our method.

3.2 Problem Settings

Basically our tracking strategy follows Markov Decision Process (MDP). The MDP is defined by states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, state transition function $s' = f(s, a)$, and the reward $r(s, a)$. In our MDP formulation, the tracker is defined as an agent of which goal is to capture the target with a bounding box shape. The action is defined in a discrete space and a sequence of actions and states is used to iteratively pursue the resulting bounding box location and size in each frame.

In every frame, the agent decides sequential actions until finalizing the target's position, and then, goes to the next frame. The state representation includes the appearance information at the bounding box of the target and the previous actions. The agent receives a reward for the final state of the frame l by deciding whether the agent succeed to track the object or not. The state and action are represented as $s_{t,l}$ and $a_{t,l}$ respectively, for $t = 1, ..., T_l$ and $l = 1, ..., L$ where T_l is the terminal step at frame l and L denotes the number of frames in a video. The terminal state in the l-th frame is transferred to the next frame, i.e., $s_{1,l+1} := s_{T_l,l}$. In the following except the sections 3.4.2 and 3.5, we omit the subscript l when we describe MDP in each frame for simplicity.

Action. The action space $\mathcal{A}$ consists of eleven types of actions including translation moves, scale changes, and stopping action as shown in Figure 3.2. The translation moves include four directional moves, $\{$*left*, *right*, *up*, *down*$\}$ and also have their two

times larger moves. The scale changes are defined as two types, $\{$*scale up*, *scale down*$\}$, which maintain the aspect ratio of the tracking target. Each action is encoded by the 11-dimensional vector with one-hot form.

State. The state s_t is defined as a tuple (p_t, d_t), where $p_t \in \mathbb{R}^{112\times112\times3}$ denotes the image patch within the bounding box (we call simply "patch" in the following) and $d_t \in \mathbb{R}^{11k}$ represents the dynamics of actions denoted by a vector (called by "action dynamics vector" in the following) containing the previous k actions at t-th iteration. The patch p_t is pointed by 4-dimensional vector

$$b_t = [x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}], \tag{3.1}$$

where $(x^{(t)}, y^{(t)})$ denotes the center position and $w^{(t)}$ and $h^{(t)}$ denote the width and height of the tracking box respectively. In a frame image F, the patch p_t at iteration t is defined as,

$$p_t = \phi(b_t, F), \tag{3.2}$$

where ϕ denotes the pre-processing function which crops the patch p_t from F at $b_t \in \mathbb{R}^4$ and resizes it to match the input size of our network. The action dynamics vector d_t is defined as concatenated past k action vectors. We store past k actions in the action dynamics vector

$$d_t = [\psi(a_{t-1}), ..., \psi(a_{t-k})], \tag{3.3}$$

where $\psi(\cdot)$ denotes one-hot encoding function. Letting $k = 10$, d_t has 110 dimension since each action vector has 11 dimension.

State transition function. After decision of action a_t in state s_t, the next state s_{t+1} is obtained by the state transition functions: patch transition function $f_p(\cdot)$ and action dynamics function $f_d(\cdot)$. The operation of the state transition function is illustrated in Figure 3.3. The patch transition function moves the location of the target according to the chosen action a_t and the action dynamics function accumulates the history of selected actions with the form of one-hot encoding.

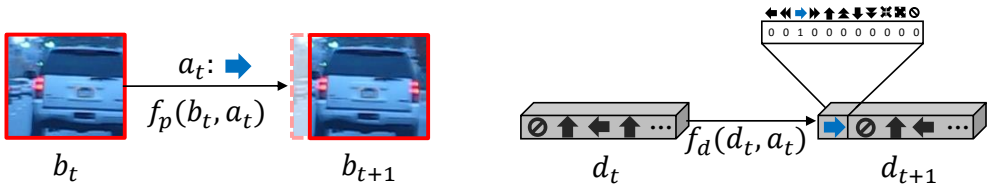


Figure 3.3: Example of state transition during action-driven tracking. **Left**: the position of the image patch b_t is translated to b_{t+1} with the *right* action. **Right**: the action dynamics vector d_t is updated to d_{t+1} by inserting the selected action.

The patch transition function is defined by

$$b_{t+1} = f_p(b_t, a_t), \tag{3.4}$$

which moves the position of the patch by the corresponding action. The discrete amount of movements is defined as

$$\begin{aligned} \Delta x^{(t)} &= \alpha w^{(t)} \\ \Delta y^{(t)} &= \alpha h^{(t)} \end{aligned} \tag{3.5}$$

where α is 0.03 in our experiments.

For example, if *'left'* action is selected, the position of the patch b_{t+1} moves to $[x^{(t)} - \Delta x^{(t)}, y^{(t)}, w^{(t)}, h^{(t)}]$ and *'scale up'* action changes the size into $[x^{(t)}, y^{(t)}, w^{(t)} + \Delta x^{(t)}, h^{(t)} + \Delta y^{(t)}]$. The other actions are defined in a similar manner. The action dynamics function is defined by $d_{t+1} = f_d(d_t, a_t)$ which represent the transition of action history. When the *'stop'* action is selected, we finalize the patch position for the target in the current frame, the agent will receive the reward, and then the resulting state is transferred to the initial state of the next frame.

Reward. The reward function is defined as $r(s)$ since the agent obtains the reward by the state s regardless of the action a. The reward $r(s_t)$ keeps zero during iteration in MDP in a frame. At the termination step T, that is, a_T is 'stop' action, $r(s_T)$ is

assigned by,

$$r(s_T) = \begin{cases} 1, & \text{if } IoU(b_T, G) > 0.7 \\ -1, & \text{otherwise,} \end{cases} \tag{3.6}$$

where $IoU(b_T, G)$ denotes overlap ratio of the terminal patch position b_T and the ground truth G of the target with intersection-over-union criterion. The tracking score z_t is defined as the terminal reward, $z_t = r(s_T)$, which will be used to update model in reinforcement learning.

3.3 Action-Decision Network Architecture

The pre-trained VGG-M model [59] is used to initialize our network. Small CNN models such as VGG-M [59] are more effective in the visual tracking problem than deep models [13]. As illustrated in Figure 3.1, our network has three convolutional layers $\{$conv1, conv2, conv3$\}$, which are identical to the convolutional layers of VGG-M networks. The next two fully connected layers $\{$fc4, fc5$\}$ are combined with the ReLU and dropout layers, and each has 512 output nodes. The output of fc5 layer is concatenated with the action dynamics vector d_t which has 110 dimensions. The final layers $\{$fc6, fc7$\}$ predict the action probabilities and confidence score of the given state respectively. The parameter of the i-th layer is denoted by w_i and the whole network parameter by W.

Table 3.1: Detail architecture of the action-decision networks. Only convolution layers and fully-connected layers are described without *ReLU* layers and *Pool* layers.

Layer	# filters	Filter size	Stride	Pad
conv1	96	$7 \times 7 \times 3$	2	0
conv2	256	$5 \times 5 \times 96$	2	0
conv3	512	$3 \times 3 \times 256$	1	0
fc4	512	$3 \times 3 \times 512$	1	0
fc5	512	$1 \times 1 \times 512$	1	0
fc6	11	$1 \times 1 \times 512$	1	0
fc7	2	$1 \times 1 \times 512$	1	0

The fc6 layer has 11 output units and is combined with softmax layer, which represents the conditional action probability distribution $p(a|s_t; W)$ for the given state. The probability $p(a|s_t; W)$ means the probability of selecting action a in the state s_t. As shown in Figure 3.1, the proposed network iteratively pursues the target position. The agent selects actions sequentially and updates states until finalizing the position of target. The final state is reached by selecting stop action or falling in the oscillation case. The oscillation case is detected, for example, when the sequential actions are obtained as $\{$*left*, *right*, *left*$\}$, which means the agent is coming back to the previous state. The confidence layer (fc7) with two output units produces the probability of target and background class for the given state s_t. The target probability $p(\text{target}|s_t; W)$ is used as the confidence score of the tracker at s_t. The confidence score is utilized for the online adaptation during tracking (Section 3.5). The detail parameters of the proposed network is described in Table 3.1.

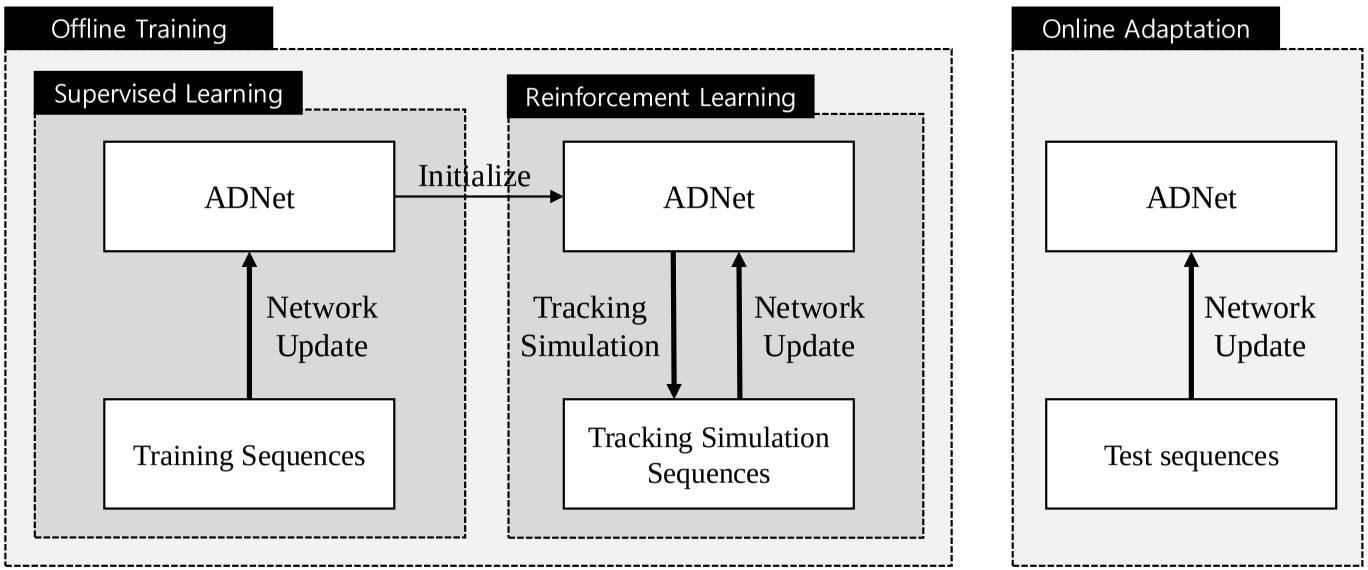


Figure 3.4: Offline and online training scheme of the action-decision network. In offline training phase, the action-decision network is first trained by supervised learning (Section 3.4.1) and then trained by reinforcement learning (Section 3.4.2). In online adaptation phase, the network is updated by the test sequences (Section 3.5).

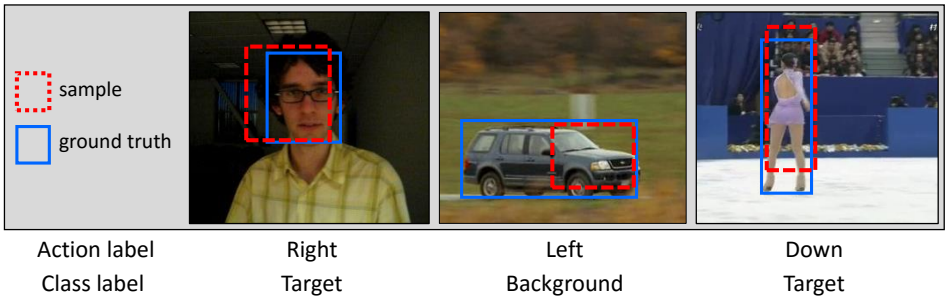


Figure 3.5: Examples of training samples for supervised learning. The red boxes and blue boxes denote the generated samples and the ground truths respectively. The action label is determined by finding the action that increases the intersection-over-union with the ground truth.

3.4 Training Methods for Action-Decision Network

In this section, we describe the training frameworks for ADNet. First, in offline manner, ADNet is pretrained by supervised learning (Section 3.4.1) and reinforcement learning (Section 3.4.2) using the training videos with the purpose of learning to track general objects. In supervised learning, the proposed network is trained to predict a proper action to a given state. In reinforcement learning, the proposed network is updated by performing tracking simulation on the training sequence and utilizing action dynamics. After pretraining ADNet, online adaptation (Section 3.5) is applied to the network to accommodate the appearance changes or deformation of the target during tracking test sequences.

3.4.1 Training ADNet with Supervised Learning

In the supervised learning stage, the network parameters W_{SL}, $\{\mathrm{w}_1, ..., \mathrm{w}_7\}$, are trained. The supervised learning process is described in Algorithm 3. We first need to generate the training samples to train ADNet (W_{SL}). The training samples consist of image

patches $\{p_j\}$, action labels $\{o_j^{(act)}\}$, and class labels $\{o_j^{(cls)}\}$. In this stage, the action dynamics is not considered and we set the elements of the action dynamics vector d_j to zero. The training datasets provide video frames and the ground truth patch position and size. Figure 3.5 illustrates the training samples generated in this stage. A sample patch p_j is obtained by adding Gaussian noise to the ground truth and the corresponding action label $o_j^{(act)}$ is assigned by,

$$o_j^{(act)} = \arg\max_a IoU(\bar{f}(p_j, a), G), \tag{3.7}$$

where G is the ground truth patch and $\bar{f}(p, a)$ denotes the moved patch from p by the action a. The class label $o_j^{(cls)}$ corresponding to p_j is defined as the following,

$$o_j^{(cls)} = \begin{cases} 1, & \text{if } IoU(p_j, G) > 0.7 \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

Algorithm 3 Training ADNet with Supervised Learning.

Input: Initialized network parameter W_{SL}, Training sequences $\{F_l\}$ and ground truths $\{G_l\}$

Output: Trained ADNet weights W_{SL}

$O^{(act)} \leftarrow \emptyset$, $O^{(cls)} \leftarrow \emptyset$

//** Generate training samples **//

for each frame F_l **do**

  Generate M randomly moved samples $\{p_1, ..., p_M\}$ using F_l and G_l

  Compute action labels $\{o_1^{(act)}, ..., o_M^{(act)}\}$ and class labels $\{o_1^{(cls)}, ..., o_M^{(cls)}\}$

  $O^{(act)} \leftarrow O^{(act)} \cup \{o_1^{(act)}, ..., o_M^{(act)}\}$, $O^{(cls)} \leftarrow O^{(cls)} \cup \{o_1^{(cls)}, ..., o_M^{(cls)}\}$

end for

//** Supervised training procedure **//

repeat

  Get m samples from $O^{(act)}$ and $O^{(cls)}$: $\{(p_j, o_j^{(act)}, o_j^{(cls)})\}_{j=1}^{m}$

  Update W_{SL} by minimizing the L_{SL} in Eq.(3.9)

until W_{SL} converges

A training batch has a set of randomly selected training samples $\{(p_j, o_j^{(act)}, o_j^{(cls)})\}_{j=1}^{m}$. ADNet ($W_{SL}$) is trained by minimizing the multi-task loss function by stochastic gradient descent. The multi-task loss function is defined by minimizing the loss L_{SL} as follows,

$$L_{SL} = \frac{1}{m}\sum_{j=1}^{m} L(o_j{}^{(act)}, \hat{o_j}{}^{(act)}) + \frac{1}{m}\sum_{i=j}^{m} L(o_j{}^{(cls)}, \hat{o_j}{}^{(cls)}), \tag{3.9}$$

where m denotes the batch size, L denotes the cross-entropy loss, and $\hat{o_j}{}^{(act)}$ and $\hat{o_j}{}^{(cls)}$ denotes the predicted action and class by ADNet, respectively.

3.4.2 Training ADNet with Reinforcement Learning

In the reinforcement learning stage, network parameters W_{RL} ($\{\mathrm{w}_1, ..., \mathrm{w}_6\}$) except fc7 layer are trained. Training ADNet with RL in this section aims to improve the network by policy gradient approach [46]. The initial RL network W_{RL} has the same parameters of the network trained by SL (W_{SL}). The action dynamics d_t is updated in every iteration by accumulating the recent k actions and shifting them in first-in-first-out strategy. Since the purpose of RL is to learn the state-action policy, we ignore the confidence layer fc7, which is needed in tracking phase.

The detailed algorithm to train ADNet with reinforcement learning is described in Algorithm 4. During the training iterations, we first randomly pick a piece of training sequence $\{F_l\}_{l=1}^{\mathcal{L}}$ and the ground truths $\{G_l\}_{l=1}^{\mathcal{L}}$ (line 3 in Algorithm 4). We then perform the reinforcement learning via tracking simulation with the training image sequences annotated by ground truth (line 4-9 in Algorithm 4). In frame l, the TRACKER iteratively pursues the position of the target by selecting actions and updating the states from the initial state $b_{T_{l-1},l-1}$ and $d_{T_{l-1},l-1}$ as shown in Algorithm 5. A tracking simulation can generate a set of sequential states $\{s_{t,l}\}$, the corresponding actions $\{a_{t,l}\}$, and the rewards $\{r(s_{t,l})\}$ for the time steps $t = 1, ..., T_l$ and frame indices $l = 1, ..., \mathcal{L}$.

Algorithm 4 Training ADNet with reinforcement learning (RL).

Input: Pre-trained ADNet (W_{SL}), Training sequences $\{F_l\}$ and ground truths $\{G_l\}$

Output: Trained ADNet weights W_{RL}

1: Initialize W_{RL} with W_{SL}

2: **repeat**

3: Randomly select $\{F_l\}_{l=1}^{\mathcal{L}}$ and $\{G_l\}_{l=1}^{\mathcal{L}}$

4: Set initial $b_{1,1} \leftarrow G_1$

5: Set initial $d_{1,1}$ as zero vector

6: $T_1 \leftarrow 1$

7: **for** $l = 2$ to $\mathcal{L}$ **do**

8: $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l \leftarrow \textsc{Tracker}(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$ in Algorithm 5

9: **end for**

10: Compute tracking scores $\{z_{t,l}\}$ with $\{b_{t,l}\}$ and $\{G_l\}_{l=1}^{\mathcal{L}}$

11: $\Delta W_{RL} \propto \sum_{l=1}^{\mathcal{L}} \sum_{t=1}^{T_l} \nabla_{W_{RL}} \log p(a_{t,l}|s_{t,l}; W_{RL}) z_{t,l}$ by Eq. (3.15)

12: Update W_{RL} using ΔW_{RL}

13: **until** W_{RL} converges

The action $a_{t,l}$ for the state $s_{t,l}$ is assigned by,

$$a_{t,l} = arg\max_{a} p(a|s_{t,l}; W_{RL}), \tag{3.10}$$

where $p(a|s_{t,l}; W_{RL})$ denotes the probability distribution of possible actions produced by the proposed ADNet for the given conditional state $s_{t,l}$. When the tracking simulation is done, tracking scores $\{z_{t,l}\}$ are computed with the ground truths $\{G_l\}$ (line 10 in Algorithm 4). In line 11-12 in Algorithm 4, the score in the tracking simulation $z_{t,l} = r(s_{T_l,l})$ is the reward at the terminal state, which obtains $+1$ for tracking success and -1 for failure at frame l, as defined in Eq. (3.6).

Algorithm 5 Tracking Procedure of ADNet.

procedure TRACKER($b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l$)
  $t \leftarrow 1$
  $p_{t,l} \leftarrow \phi(b_{T_{l-1},l-1}, F_l)$
  $d_{t,l} \leftarrow d_{T_{l-1},l-1}$
  $s_{t,l} \leftarrow (p_{t,l}, d_{t,l})$
  repeat
    $a_{t,l} \leftarrow arg\max_a p(a|s_{t,l}; W)$
    $b_{t+1,l} \leftarrow f_p(b_{t,l}, a_{t,l})$
    $p_{t+1,l} \leftarrow \phi(b_{t+1,l}, F_l)$
    $d_{t+1,l} \leftarrow f_d(d_{t,l}, a_{t,l})$
    $s_{t+1,l} \leftarrow (p_{t+1,l}, d_{t+1,l})$
    $t \leftarrow t + 1$
  until $s_{t,l}$ is a terminal state
  Set termination step $T_l \leftarrow t$
  Return $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l$
end procedure

To define the objective to train the ADNet, we first denote the tracking simulation sequences $\{u_l\}_l^{\mathcal{L}}$, where $u_l = \{s_{t,l}, a_{t,l}\}_{t=1}^{T_l}$ is the state and action sequences at frame l. The tracking performance of u_l is denoted as $R(u_l) = \frac{1}{T_l}\sum_t^{T_l} z_{t,l} = z_{T_l,l}$. The goal of the agent is to maximize the tracking performance under the distribution $p(u_l; W_{RL})$:

$$\begin{aligned} J(W_{RL}) &= E_{u_l \sim p(u_l; W_{RL})}[R(u_l)] \\ &= \sum_{u_l \in \mathbb{U}} p(u_l; W_{RL}) R(u_l), \end{aligned} \tag{3.11}$$

where $\mathbb{U}$ denotes the set of all the possible sequences and $p(u_l; W_{RL})$ is the distribution over the tracking sequence u_l depending on the policy at l-th frame. Using the Markov assumption, the expansion of $p(u_l; W_{RL})$ is given by,

$$p(u_l; W_{RL}) = p(s_{1,l})\Pi_{t=1}^{T} p(s_{t+1,l}|s_{t,l}, a_{t,l}) p(a_{t,l}|s_{t,l}; W_{RL}). \tag{3.12}$$

Taking the gradient of (3.11) to maximize the objective J,

$$\nabla_{W_{RL}} J(W_{RL}) = \sum_{u_l \in U} \nabla_{W_{RL}} p(u_l; W_{RL}) R(u_l). \tag{3.13}$$

Using the log-likelihood trick:

$$\nabla_{W_{RL}} \log p(u; W_{RL}) = \frac{\nabla_{W_{RL}} p(u; W_{RL})}{p(u; W_{RL})},$$

(3.13) leads to

$$\begin{aligned} &\nabla_{W_{RL}} J(W_{RL}) \\ &= \sum_{u_l \in U} p(u_l; W_{RL}) \nabla_{W_{RL}} \log p(u_l; W_{RL}) R(u_l) \\ &= E_{u_l \sim p(u_l; W_{RL})} \left[\nabla_{W_{RL}} \log p(u_l; W_{RL}) R(u_l)\right]. \end{aligned} \tag{3.14}$$

Replacing $p(u_l; W_{RL})$ with (3.12) and ignoring the terms unrelated with W_{RL},

$$\begin{aligned}
&\nabla_{W_{RL}} J(W_{RL}) \\
&\propto E_{u_l \sim p(u_l; W_{RL})} \left[\sum_t \nabla_{W_{RL}} \log p(a_{t,l} | s_{t,l}; W_{RL}) R(u_l) \right] \\
&\approx \frac{1}{\mathcal{L}} \sum_{l}^{\mathcal{L}} \sum_{t}^{T_l} \nabla_{W_{RL}} \log p(a_{t,l}, s_{t,l}; W_{RL}) R(u_l),
\end{aligned}$$

where the last line is obtained from a sample approximation [46]. Since $R(u_l) = z_{T_l,l}$, therefore, the network parameters W_{RL} are updated by stochastic gradient ascent to maximize the objective function (3.11) using the approximated gradient:

$$\Delta W_{RL} \propto \sum_{l=1}^{\mathcal{L}} \sum_{t=1}^{T_l} \nabla_{W_{RL}} \log p(a_{t,l} | s_{t,l}; W_{RL}) z_{T_l,l}. \tag{3.15}$$

Our framework can train ADNet even if the ground truths $\{G_l\}$ are partially given, which means the semi-supervised setting as shown in Figure 3.6. Red boxes and blue boxes denote the ground truths and the predicted target's positions respectively. In this example, only the frames $\#160$, $\#190$, and $\#220$ are annotated. Through the sequential actions, the agent at frame $\#190$ receives $+1$ reward and -1 at frame $\#220$. Therefore, the tracking scores from frame $\#161$ to $\#190$ will be +1 and -1 between $\#191$ and $\#220$. The supervised learning framework cannot learn the information of the unlabeled frames, however, the reinforcement learning can utilize the unlabeled frames in semi-supervised manner. In order to train ADNet in RL, the tracking scores $\{z_{t,l}\}$ should be determined, but the tracking scores in the unlabeled sequences cannot be immediately determined. Instead, we assign the tracking scores to the reward obtained from the result of tracking simulation. In other words, if the result of tracking simulation during the unlabeled sequences is evaluated as success at the labeled frame, the tracking scores for the unlabeled frames are given by $z_{t,l} = +1$. If it is not successful, $z_{t,l}$ is assigned by -1, as shown in Figure 3.6.

The tracking performance during reinforcement learning is illustrated in Figure 3.7. We conduct the proposed deep reinforcement algorithm on the training videos, especially *Fernando* sequences in this example. The blue colored bounding boxes represents the position of the tracker. The first column shows the initial frame with ground truth. From the second column to the last column, we plot the position of the tracker at the final frame according to the number of iterations. When the number of iteration is zero, that is second column, ADNet is the same as the trained network by supervised learning and shows poor tracking results. We can see the tracking performance is improved as training iteration increases.

3.5 Online Adaptation in Tracking

The proposed network is updated in online manner during tracking. This online adaptation can make the tracking algorithm more robust against the appearance changes or deformation. When updating ADNet, we fix the convolutional filters $\{\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3\}$ and fine-tune the fully-connected layers $\{\mathrm{w}_4, ..., \mathrm{w}_7\}$ because the convolutional layers would have generic tracking information whereas the fully-connected layers would have the video-specific knowledge.

The detailed procedure of the proposed tracking and online adaptation is described in Algorithm 6. The tracking is performed by deciding sequential actions with the state-action probability $p(a|s; W)$. We adopt the online update adaptation of [13]. The online adaptation is done by fine-tuning ADNet with a supervised learning using the temporal training samples generated during the tracking process. For the supervised learning, training samples with labels are required. For the labeling, we have to determine the ground truth. The tracked patch position determined by the network is used for the temporal ground truth. As similar to SL (Section 3.4.1), the training sample set $\mathbb{S}$ for online adaptation consists of image patches $\{p_i\}$ sampled randomly around

Table 3.2: Parameters for ADNet and ADNet-fast.

	ADNet	ADNet-fast
Initial samples # ($\mathcal{N}_I$)	3000	300
Online samples # ($\mathcal{N}_O$)	250	50
Re-detection samples # ($\mathcal{N}_{det}$)	256	64
Online adaptation in every ($\mathcal{I}$) frames	10	30

the tracked patch position and the corresponding action labels $\{o_i^{act}\}$ and class labels $\{o_i^{cls}\}$. The labels are obtained via Eq. (3.7) and Eq. (3.8). At the first frame, the initial samples $\mathbb{S}_{init}$ are generated using the initial target position, and ADNet is fine-tuned to fit the given target. At frame $l(\geq 2)$, the training samples $\mathbb{S}_l$ are generated using the tracked patch position $b_{T_l,l}$ if the confidence score $c(s_{t,l})$ of the estimated target is above 0.5. The confidence score $c(s_{t,l})$ of the state $s_{t,l}$ is defined as the target probability $p(\text{target}|s_{t,l}; W)$ of the confidence layer (fc7). Online adaptation is conducted for every $\mathcal{I}$ frames using the training samples $\{\mathbb{S}_k\}_{k=l-\mathcal{J}+1}^{l}$, which means the online adaptation uses the training samples generated from the past $\mathcal{J}$ frames. When the score $c_{t,l}$ is below -0.5, which means the tracker miss the target, then the *re-detection* is conducted to capture the missed target. The target position candidates $\{\tilde{b}_i\}_{i=1}^{\mathcal{N}_{\text{det}}}$ are generated around the current target position with random Gaussian noise. The re-detected target position b^* is selected by

$$b^* = arg\max_{\tilde{b}_i} c(\tilde{b}_i), \tag{3.16}$$

and the state $s_{T_l,l}$ is assigned by the target position b^* and the action dynamics vector $d_{T_l,l}$.

3.6 Implementation Details

3.6.1 Pretraining the ADNet

In each frame, we generated the training samples $\{p_j\}$ from Gaussian distribution whose mean is zero and covariance matrix is diag$((0.3w)^2, (0.3h)^2, (0.1w)^2, (0.1h)^2)$. When pretraining ADNet, we draw 250 samples in each frame. We set the learning rate to 0.0001 for convolutional layers (fc1-3) and 0.001 for fully-connected layers (fc4-7) [13], momentum to 0.9, weight decay to 0.0005, and mini-batch size to 128. For pretraining ADNet with $\mathcal{K}$ training videos, the number of training iteration is set to 300 for each video. In each iteration of the reinforcement learning, we randomly select the sequence of length $\mathcal{L}(= 10)$ for the tracking simulation.

3.6.2 Online Adaptation of the ADNet

For online adaptation, we only train the fully-connected layers (fc4-7) with the learning rate 0.001. In addition to the normal version (ADNet), to reduce the computation in actual tracking, we can apply the fast version of ADNet using a small number of samples in online adaptation, referred to as "ADNet-fast". The parameters of online adaptation for ADNet and ADNet-fast are described in Table 3.2. The ADNet is fine-tuned with $\mathcal{T}_I(= 300)$ iterations at the first frame and $\mathcal{T}_O(= 30)$ iterations for the online adaptations. The online training is conducted for every $\mathcal{I}(= 10)$ frames and the training data are sampled from the past $\mathcal{J}(= 20)$ frames. For the *re-detection*, we draw $\mathcal{N}_{\text{det}}(= 256)$ target position candidates. In the online adaptation, $\mathcal{N}_{\text{I}}(= 3000)$ samples are generated in the first frame, and $\mathcal{N}_{\text{O}}(= 250)$ samples are generated in the frame whose confidence is above 0.5 during tracking.

In ADNet-fast, we set $\mathcal{N}_{\text{I}}$ to 300, $\mathcal{N}_{\text{O}}$ to 50, $\mathcal{I}$ to 30, and $\mathcal{N}_{\text{det}}$ to 64. Tracking with ADNet-fast endures 3% performance degradation but achieves a real time speed around 4 times faster than the standard version of ADNet.

Figure 3.6: Illustration of the tracking simulation in semi-supervised case on *Walking2* sequence. In semi-supervised case, the ground truth position is provided only at the end of the sequence, therefore, the agent receives the reward when the tracking simulation is done. **Top:** tracking simulation is conducted from frame #160 to #190. The agent receives positive reward as the tracking simulation is successful. **Bottom:** tracking simulation is conducted from frame #190 to #220. The agent receives negative reward since the agent missed the target.

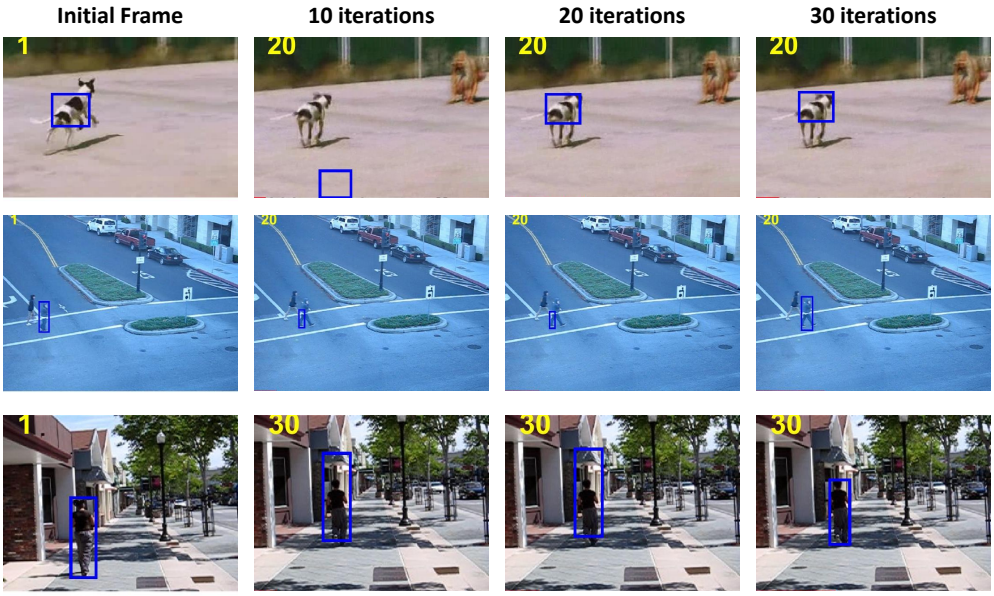


Figure 3.7: Tracking performance during reinforcement learning for the tracking simulation sequences. We can see that the tracking results are improved as training iteration progresses.

Algorithm 6 Online adaptation of ADNet in tracking.

Input: Trained ADNet (W), Test sequences $\{F_l\}_{l=1}^{L}$ and initial target position $b_{1,1}$

Output: Estimated target positions $\{b_{T_l,l}\}_{l=2}^{L}$

Generate samples $\mathbb{S}_1$ from F_1 and G_1
Update W using $\mathbb{S}_1$
Set initial $d_{1,1}$ as zero vector
$T_1 \leftarrow 1$
for $l = 2$ to L **do**
    $\{a_{t,l}\}, \{b_{t,l}\}, \{d_{t,l}\}, T_l \leftarrow \text{TRACKER}(b_{T_{l-1},l-1}, d_{T_{l-1},l-1}, F_l)$ in Algorithm 5
    $s_{T_l,l} \leftarrow (\phi(b_{T_l,l}, F_l), d_{T_l,l})$
    if $c(s_{T_l,l}) > 0.5$ **then**
        Generate samples $\mathbb{S}_l$ from F_l and $b_{T_l,l}$
    else if $c(s_{T_l,l}) < -0.5$ **then**
        Draw $\mathcal{N}_{\text{det}}$ target position candidates $\{\tilde{b}_i\}$
        Re-detect the target position $b^* \leftarrow arg\max_{\tilde{b}_i} c(\tilde{b}_i)$
        $s_{T_l,l} \leftarrow (\phi(b^*, F_l), d_{T_l,l})$
    end if
    if $\text{mod}(l, \mathcal{I}) = 0$ **then**
        Update W using the samples $\{\mathbb{S}_k\}_{k=l-\mathcal{J}+1}^{l}$
    end if
end for

Chapter 4

Interacted Action-Driven Visual Tracking

4.1 Overview

In the previous chapter, we proposed action-driven method for visual object tracking. The proposed action-decision network, ADNet, aims to learn to track a target object by selecting proper actions according to the states of the target object. However, it is still very challenging to track the target object when the scene is complex due to severe occlusions by other similar objects. Figure 4.1 shows examples of complex situation such as sports scene or crowded scene. The ADNet could not consider the *global* situation of the scene, since it takes the local patch feature around the target regardless the status of the other objects. In the complex scene, it is difficult to understand and track the movement of the target in a local view because the objects interact with each other and has complicated moving patterns.

Figure 4.2 shows the failure case due to the interference from other object which has a similar appearance. The tracker get confused when two similar objects are over-

Figure 4.1: Example of complex and crowded tracking scene.

lapped, therefore, the tracker decides to track the other object. In order to solve this problem, we develop a multi-agent architecture to consider interactions among the target object and its neighboring objects. The proposed multi-agent architecture refers to multi-agent action-decision network (ADNet) in the following. To consider the interaction of objects in a complex scene, global context-aware tracking methods have been proposed for the purpose of single object tracking [16, 17] and multiple object tracking [18, 19]. In the proposed multi-agent ADNet, the action-decision network works as an agent for single object tracking and multiple agents for neighboring objects are interacting with others.

The proposed multi-agent ADNet is depicted in Figure 4.3. Instead of considering only the state of the target agent, the multi-agent ADNet learns the policy that determines an action by considering the states of neighboring agents at the same time. For the interacted policy learning, an inter-agent communication channel and an additional policy network are added as shown in Figure 4.3. We propose a novel network architecture to jointly learn inter-agent policy. To train the multi-agent ADNet, we adopt a similar strategy to the single-agent action-decision network. At the testing phase, we also adopt the online adaptation scheme (Section 3.5) to obtain robust performance and

Figure 4.2: Failure to track due to interference from other object with similar appearance. The red box and blue box denote the tracking bounding box and the ground truth respectively.

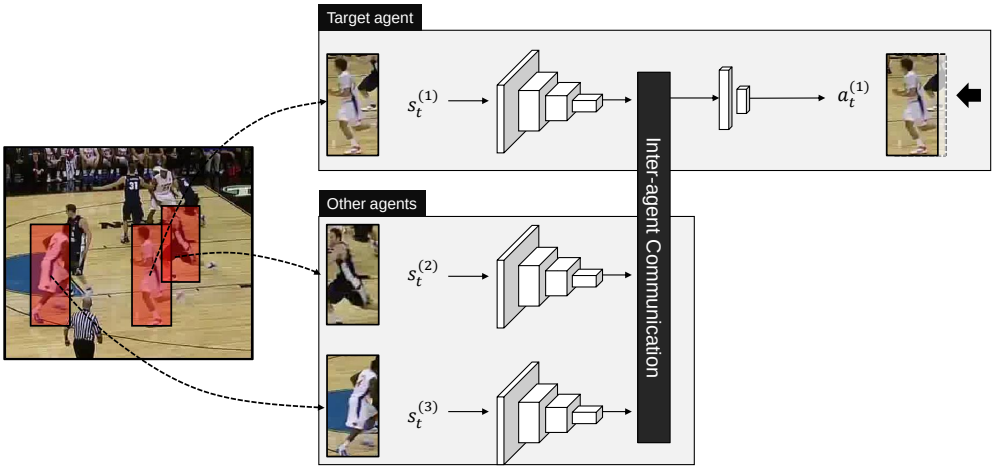


Figure 4.3: The overall scheme of multi-agent ADNet. For the convenience of explanation, the figure only presents action-decision process for the target agent.

prevent drift problem. The experiments shows that the proposed multi-agent ADNet achieves better performance on the challenging sports sequence than the single-agent version of our method and the state-of-the-art trackers.

In Seciton 4.2, we define the problem of visual tracking with interactions. Seciton 4.3 presents the detailed description of our method. Seciton 4.4 describes the training method utilizing supervised learning and reinforcement learning.

4.2 Problem Settings

The problem we are trying to solve is to localize the target objects in sequential frames. In the given frame, several objects are interact with each other by a specific purpose. In our settings, each individual agent is initialized at the start frame of the object. After initializing every agent, the tracking process is performed to subsequent frames.

Here we denote notations used for multi-agent ADNet.

- (i): agent index
- $(-i)$: agents except the agent i.
- $s_t^{(i)}$: state of agent i at time step t composed of patch and action dynamics.
- $a_t^{(i)}$: selected action of the agent i.
- $o_t^{(-i)}$: embedded message feature from the states $s^{(-i)}$ of other agents with respect to the geometric relationship between $s^{(i)}$ by the *message encoder*. Details are described in Section 4.3.2.1.
- $\phi(\cdot)$: feature encoder for state s with convolutional neural networks, which is the same as $\{$conv1-3 and fc4-5$\}$ layers of ADNet.
- $f(\cdot)$: fully-connected layers connected to $|\mathcal{A}|$-dim output nodes for the feature of target state $s^{(i)}$, which is the same as $\{$fc6-7$\}$ layers of ADNet.

- $h(\cdot)$: fully-connected layers connected to $|\mathcal{A}|$-dim output nodes for the message $o^{(i)}$.
- $q_t^{(i)}$, $q_t^{(-i)}$: $|\mathcal{A}|$-dim vector coming out of f and h respectively, to determine the action $a_t^{(i)}$.
- $g(\cdot)$: fully-connected layers outputs 2-dim vector with softmax layer. Details are described in Section 4.3.2.2.
- $v_t^{(i)}$, $v_t^{(i)}$: outputs of the *selector* module.
- $p(a_t^{(i)}|s_t^{(i)}, o_t^{(-i)}; \theta)$: **policy** parameterized by θ, the state-action probability distribution conditioned on the target state $s_t^{(i)}$ and the received message $o_t^{(-i)}$ from the other agents.
- $z_t^{(i)}$: return of agent i, in other word, tracking score of agent (i) at time t used in reinforcement learning.

We use the same definition of the state, action, state transition function, and reward in Section 3.2. The state and action are represented as $s_{t,l}$ and $a_{t,l}$ respectively, for $t = 1, ..., T_l$ and $l = 1, ..., L$ where T_l is the terminal step at frame l which is the same as Section 3.2. We omit the subscript l in this chapter, which were also omitted in Section 3.2. The **state** is composed of 112×112 image patch and 110-dim action dynamics vector. We omit the representation of action dynamics in the figures of this chapter for simplicity.

4.3 Proposed Method

4.3.1 Baseline

To extend from single-agent method presented in Chapter 3 to multi-agent system, we first describe the baseline framework of multi-agent ADNet. The baseline of the pro-

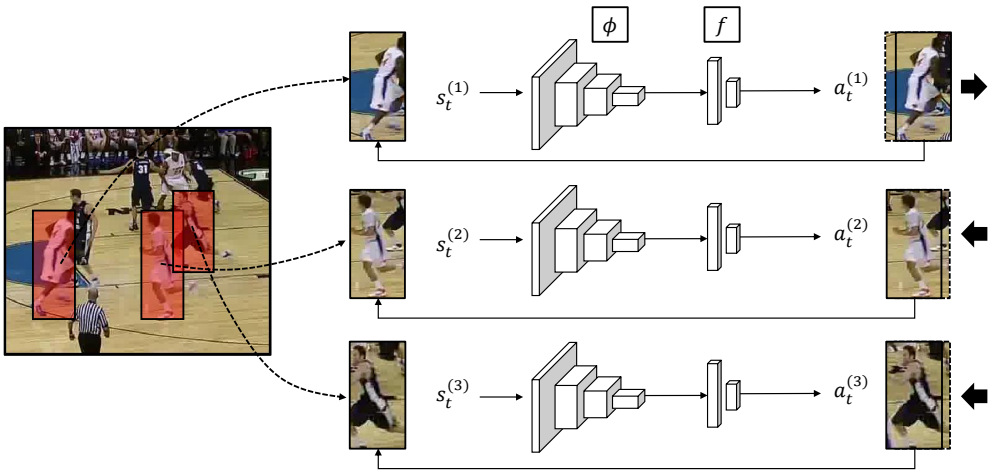


Figure 4.4: Baseline independent action-decision network. For the convenience of the explanation, the *action dynamics* are omitted in this figure. In real implementation the state $s^{(i)}$ of agent (i) has an image patch and action dynamics feature.

posed method is *independent ADNet*, which is inspired by independent-DQN [55]. In the independent-DQN method, each agent i independently and simultaneously learns its own Q-function $Q^i(s, a^i; \theta^i)$. Similarly, the *independent ADNet* independently determines the action of each agent according to its state, however, the learning algorithm is the same as in the single-agent case since the interaction among agents is not specified as in the independent-DQN.

The baseline framework is depicted in Figure 4.4. For the convenience of the explanation, the *action dynamics* are omitted in this figure. In real implementation the state $s^{(i)}$ of agent (i) has an image patch and action dynamics feature. As shown in Figure 4.4, each state $s_t^{(i)}$ at time step t is fed into the action-decision network to decide the proper action $a_t^{(i)}$. Then, by the state-transition function, the state is moved to the next state $s_{t+1}^{(i)}$.

The independent ADNet architecture does not have an interaction module among

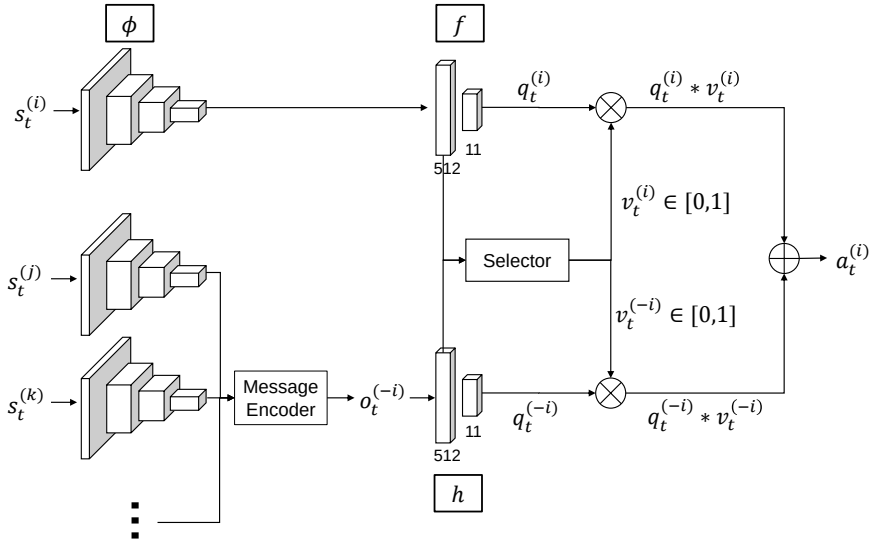


Figure 4.5: Inter-agent communication scheme and interacted policy network structure.. For the convenience of the explanation, the *action dynamics* are omitted in this figure.

agents. Each agent selects a proper action to capture the target considering only the local state, appearance from the image patch and motion information from the action dynamics. This method successfully performs tracking in situations where the target objects do not overlap, but tracking often fails when the target object is interfered by objects with similar appearance. Therefore, we propose a method to learn the policy of selecting actions by considering the states to other objects at the same time.

Table 4.1: Detail structure of the multi-agent ADNet. Only convolution layers and fully-connected layers are described without *ReLU* layers and *Pool* layers.

Layer	# filters	Filter size	Stride	Pad
ϕ-conv1	96	$7 \times 7 \times 3$	2	0
ϕ-conv2	256	$5 \times 5 \times 96$	2	0
ϕ-conv3	512	$3 \times 3 \times 256$	1	0
ϕ-fc4	512	$3 \times 3 \times 512$	1	0
ϕ-fc5	512	512	1	0
f-fc1	11	512	1	0
f-fc2	2	512	1	0
h-fc	11	512	1	0
Message-fc	512	$3 \times 3 \times 512$	1	0
Selector-fc1	512	1024	1	0
Selector-fc2	2	512	1	0

4.3.2 Multi-agent Architecture

The proposed multi-agent architecture is depicted in Figure 4.5. The figure shows only the parts to determine the action of agent i. We denote the other agents, such as j or k, as $(-i)$. The input of the network is the set of states at time t, $\{s_t^{(1)}, ..., s_t^{(i)}, ..., s_t^{(N)}\}$, and the output is the state-action distribution as following,

$$p(a_t^{(i)}|s_t^{(1)}, ..., s_t^{(i)}, ..., s_t^{(N)}; \theta) = p(a_t^{(i)}|s_t^{(i)}, o_t^{(-i)}; \theta), \tag{4.1}$$

where $o_t^{(-i)}$ denotes the message from the other agents $s_t^{(-i)}$.

All the states, ($s_t^{(i)}$, $s_t^{(j)}$, $s_t^{(k)}$, ...), are fed into the shared convolutional layers ϕ, which is the same structure as $\{$conv1-3,fc4-5$\}$ of ADNet. The convolutional feature encoder ϕ outputs 512-dimensional feature vector and connected to 11-dimensional output nodes through fully-connected layers f. The 11-dimensional output vector represents *primitive* action $q_t^{(i)}$.

To aggregate the information of other agents, the *message encoder* module (Section 4.3.2.1) is introduced. The message encoder outputs 512-dimensional aggregated message $o_t^{(i)}$ and the message is fed into fully connected layers h which outputs 11-dimensional output vector. The 11-dimensional output vector of h represents *primitive* action for the other agents $q_t^{(-i)}$.

In the single-agent ADNet, $q_t^{(-i)}$ is directly defined as the state-action probability $p(a|s;\theta)$. In contrast, the multi-agent ADNet considers both $q_t^{(i)}$ and $q_t^{(-i)}$ to obtain the state-action probability. In order to combine the two primitive actions, the action *selector* module (Section 4.3.2.2) is proposed. The selector has 2-dimensional output nodes $v_t^{(i)}$ and $v_t^{(-i)}$, where each value is multiplied with each primitive action and added to obtain the final action distribution $a_t^{(i)}$.

$$a_t^{(i)} = v_t^{(i)} q_t^{(i)} + v_t^{(-i)} q_t^{(-i)}, \tag{4.2}$$

where $v_t^{(i)}$ and $v_t^{(-i)}$ are in range $[0, 1]$. The action of agent i at time t is determined by

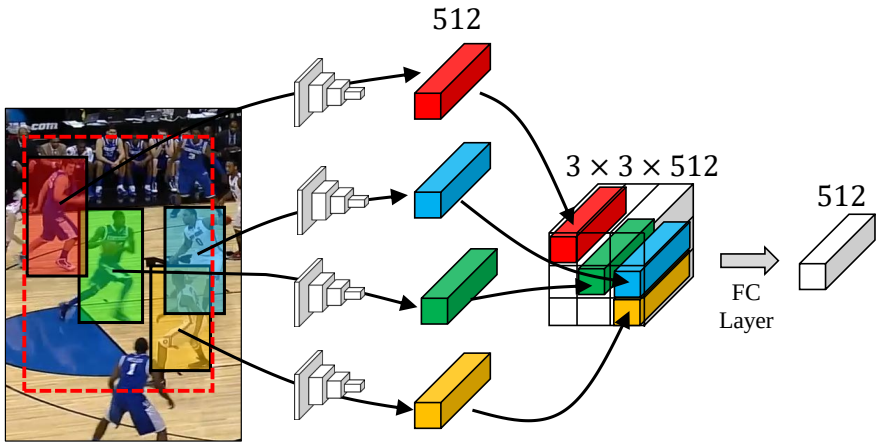


Figure 4.6: Message encoder to aggregate messages from neighboring agents. The target tracker is denoted by green bounding box and the region-of-interest is represented as the dashed red line. The neighboring agents are represented as red, blue, and yellow bounding boxes. Each state is encoded by ϕ and aligned into 3×3 grid. Then a messages from fully connected layer is connected to output layer yielding 512-dimensional message vector.

the above process, and the actions of other agents are also determined in the same way. That is, all the network parameters of feature encoder, message encoder, and selector are shared by all agents.

Table 3.1 shows the detailed structure of the proposed multi-agent ADNet. The feature encoder ϕ has 5 layers ϕ-{conv1-3,fc4-5}, f has 2 layers f-{fc1,fc2}, h has a layer h-fc, the message encoder has a layer **Message**-fc , and the action selector has two layers **Selector**-{fc1,fc2}. The f-{fc2} has the same operation as fc7 layer of single-agent ADNet (Section 3.4.1) to obtain the confidence score of the tracking status, which is used during online tracking phase.

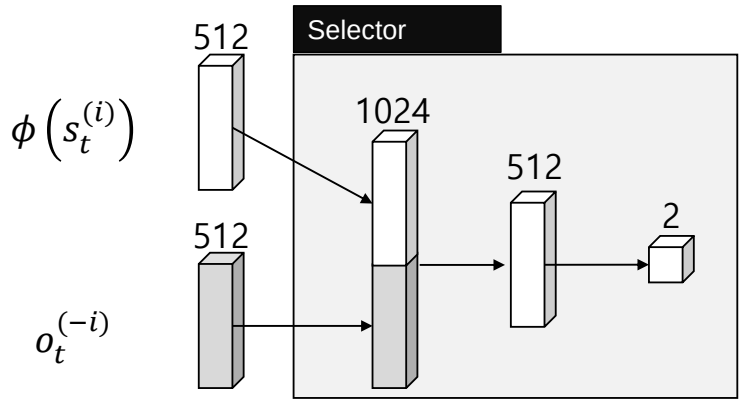


Figure 4.7: Action selector of multi-agent ADNet.

4.3.2.1 Message Encoder

The proposed message encoder is presented in Figure 4.6. As shown in the figure, the 512-dimensional features are computed using ϕ for each state. In message encoder, we only consider the agents within the region-of-interest (ROI) of the target agent since the target agent has little interaction with far away agents. The ROI is obtained by expanding the size of the target bounding box by 1.5 times. The encoded feature using ϕ of the agents inside the ROI is aligned to the 3×3 grid considering the geometric relation with the target. If there are several agents in a grid area, the closest one with the target is chosen. The aligned feature is 3 tensor and connected to fully connected layer (**Message**-fc) which outputs the encoded message $o_t^{(i)}$.

4.3.2.2 Action Selector

The action selector module is depicted in Figure 4.7. The action selector determines the proper action between the primitive actions $q_t^{(i)}$ and $q_t^{(-i)}$. Since we need to make decisions based on the overall situation, the action selector takes two-way inputs of

$\phi(s_t^{(i)})$ and $o_t^{(-i)}$ for the target state and the neighbor states respectively. The inputs $\phi(s_t^{(i)})$ and $o_t^{(-i)}$ are concatenated to 1024-dimensional vector and the output is 2-dimensional vector through two fully connected layer (**Selector**-{fc1-2}) and softmax layer.

Therefore, the 2-dimensional output, $v_t^{(i)}$ and $v_t^{(-i)}$, have values between 0 and 1 with a sum of 1. The output values $v_t^{(i)}$ and $v_t^{(-i)}$ are used to determine the final action as Eq.(4.2). The primitive actions, $q_t^{(i)}$ and $q_t^{(-i)}$, are multiplied with the output values, $v_t^{(i)}$ and $v_t^{(-i)}$, respectively.

4.4 Training Methods for Multi-agent ADNet

The proposed training method consists of two-step process that first pretrains the network weights using supervised learning and fine-tunes using reinforcement learning as in the single-agent ADNet (Section 3.4). We describe the supervised Learning method in Section 4.4.1 and the reinforcement learning method in Section 4.4.2.

4.4.1 Training Multi-agent ADNet with Supervised Learning

Even though end-to-end learning of the multi-agent ADNet is possible, but we train the network with three steps: learning of object features, learning for message encoder, and learning for action selector.

Here we describe the training method of ϕ and f. The network layers of ϕ and f in Multi-agent ADNet are the same as single-agent ADNet, so we train ϕ and f with the same way in Section 3.4.1. That is, the network is trained to learn action $q_t^{(i)}$ for input state $s_t^{(i)}$ with supervised manner. In this process, agent index (i) is omitted because the relationship between agents is not considered. In addition, f-fc2 layer is trained in this step, which computes the confidence score for the tracking status. f-fc2 layer has the same role as fc7 in Section 3.4.1. The training samples $\{s_j, o_j^{(act)}, o_j^{(cls)}\}$ are

obtained by translating and scale jittering the ground truths. We denote s_j, $o_j^{(act)}$ and $o_j^{(cls)}$ as the input state, ground truth action, and the ground truth class label. The loss function for feature encoding, L_{FE}, is

$$L_{FE} = \frac{1}{m}\sum_{j=1}^{m} L(o_j^{(act)}, \hat{o}_j^{(act)}) + \frac{1}{m}\sum_{j=1}^{m} L(o_j^{(cls)}, \hat{o}_j^{(cls)}), \tag{4.3}$$

where $\hat{o}_j^{(act)}$ and $\hat{o}_j^{(cls)}$ are the predicted action and class label obtained by f-fc and f-fc2, respectively.

4.4.1.1 Training of Message Encoder

In this step, the fully-connected layers, **Message**-fc and h, are trained while the parameters of ϕ and f are freezing. The training samples are denoted by $\{n_j, y_j^{(act)}\}$, where n_j ($\in \mathbb{R}^{3\times3\times512}$) denotes the aligned feature of neighboring agents and $y_j^{(act)}$ the ground truth action. The ground truth action $y_j^{(act)}$ is obtained by selecting the action that yields to maximize IoU with the ground truth position from the previous ground truth position. The loss function for message encoding, L_{ME}, is

$$L_{ME} = \frac{1}{m}\sum_{j=1}^{m} L(y_j^{(act)}, \hat{y}_j^{(act)}), \tag{4.4}$$

where $\hat{y}_j^{(act)}$ is the predicted action of j-th agent and m is the size of training minibatch.

4.4.1.2 Training of Action Selector

While training the weights of selector layers **Selector**-fc1 and **Selector**-fc2, the pre-trained layers ϕ, f, h, and **Message**-fc are freezing. Training samples are $\{s_i, n_i, y_i^{(sel)}\}$, where s_i and n_i are the target agent state and neighboring agent state. The ground truth label $y_i^{(sel)}$ is assigned to 1 when the primitive action of the target state is selected, that is, the action predicted from target state has better IoU with ground truth position than the action predicted from the neighboring states.

The loss function for action selector, L_{Sel}, is

$$L_{Sel} = \frac{1}{m}\sum_{j=1}^{m} L(y_j^{(sel)}, \hat{y}_j^{(sel)}), \tag{4.5}$$

where $\hat{y}_j^{(sel)}$ is predicted selection and m is the size of training minibatch.

4.4.2 Training of Multi-agent ADNet with Reinforcement Learning

After training the supervised learning, the network is fine-tuned by reinforcement learning. The detail algorithm is described in Algorithm 7. The weights for multi-agent ADNet, including ϕ, f, h, **Message**-fc, and **Selector**-$\{$fc1-2$\}$, are initialized with the pretrained weights. At each tracking simulation, a tracking sequences of length $\mathcal{L}$ is randomly selected. Through tracking simulation, the states and actions can be obtained by the policy $p(a|s, o; \theta)$ for each agent. After the tracking simulation is done, the tracking score $z_{t,l}^{(i)}$ is computed by Eq.(3.6). The network weights θ is updated by policy gradient [46] as following,

$$\Delta\theta \propto \sum_{l=1}^{\mathcal{L}}\sum_{i=1}^{N_a}\sum_{t=1}^{T_l} \nabla_\theta \log p\left(a_{t,l}^{(i)}|s_{t,l}^{(i)}, o_{t,l}^{(-i)}; \theta\right) \cdot z_{t,l}^{(i)}. \tag{4.6}$$

4.5 Online Adaptation in Tracking

The multi-agent ADNet is updated in online manner during tracking as similar in Section 3.5. The online adaptation can make the tracking algorithm more robust against the problem of illumination change or deformation. The action-decision layers f aim to choose the proper action to capture the target by the appearance and the action dynamics. Since message encoder and the action selector aim to understand the interaction among agents, the online adaptation updates the action-deciding layers of the target agent f while the other layers are freezing.

At the initial frame, the ground truths of all the tracking targets are given. Then training samples are generated using the initial target position and the feature extractor f of each agent is fine-tuned using the training samples. After the initialization, tracking is performed by the proposed multi-agent ADNet tracker. When the confidence score of the estimated target state is above 0.5, the training samples are stored to update the target tracker similarly to Section 3.5. For every 10 frames, the online adaptation is conducted using the stored samples. Also, re-detection procedure is also performed to recover the lost target using Eq.(3.16). Parameters used in online adaptation are set to the same as single-agent ADNet.

4.6 Implementation Details

4.6.1 Pretraining

At pretraining phase, the ground truth bounding boxes of the target for all frames are given. In each frame, we generate the training samples to train the network with supervised manner. The training samples are generated using Gaussian distribution whose mean is zero and covariance matrix is diag$((0.3w)^2, (0.3h)^2, (0.1w)^2, (0.1h)^2)$ by adding to the ground truth position. When pretraining the network, we draw 250 samples in each target. We set the learning rate to 0.0001 for the convolutional layers of ϕ and 0.001 for fully-connected layers of ϕ. The proposed network is trained by stochastic gradient descent method and we set the momentum to 0.9, weight decay to 0.0005, and mini-batch size to 128.

4.6.2 Online Adaptation

For online adaptation, the fully-connected layers of ϕ for each target are only trained with learning rate 0.001. The network is fine-tuned with 300 iterations at the first frame and 30 iterations for the online adaptations. The online adaptation is conducted for

every 10 frames and the training data are sampled from the stored samples during the past 20 frames. For the *re-detection*, we draw 256 target position candidates. In the online adaptation, 3000 samples are used in the first frame, and 250 samples are used in the frame whose confidence is above 0.5 during tracking.

Algorithm 7 Training multi-agent ADNet with reinforcement learning.

Input: Pre-trained multi-agent ADNet θ_{SL}, Training sequences $\{F_l\}$ and ground truths $\{G_l^{(i)}\}$

Output: Trained multi-agent ADNet weights θ

Initialize θ with θ_{SL}

repeat

// Tracking simulation

Randomly select a sequence $\{F_l\}_{l=1,...,\mathcal{L}}$ and ground truths $\{G_l^{(i)}\}_{l=1,...,\mathcal{L}}^{i=1,...,N_a}$ for $l = 1, ..., \mathcal{L}$

for frame index $l = 1$ to $\mathcal{L}$ **do**

for agent $i = 1$ to N_a **do**

if l is 1 **then**

Set initial state $s_{1,l}^{(i)}$ using $G_1^{(i)}$

else

$s_{1,l}^{(i)} \leftarrow s_{T_{l-1},l-1}^{(i)}$

$\{s_{t,l}^{(i)}\}_{t=1,...,T_l}, \{a_{t,l}^{(i)}\}_{t=1,...,T_l} \leftarrow$ MATRACKER $(s_{1,l}^{(i)}, s_{1,l}^{(-i)}, F_l)$

end if

end for

end for

// Compute tracking scores

Compute $\{z_{t,l}^{(i)}\}$ with $\{s_{t,l}^{(i)}\}$ and $\{G_l^{(i)}\}$ for all agents $i = 1, ..., N_a$

// Updating weights

$\Delta\theta \propto \sum_{l=1}^{\mathcal{L}} \sum_{i=1}^{N_a} \sum_{t=1}^{T_l} \nabla_\theta \log p\left(a_{t,l}^{(i)} | s_{t,l}^{(i)}, o_{t,l}^{(-i)}; \theta\right) \cdot z_{t,l}^{(i)}$

until θ converges

Algorithm 8 Tracking Procedure of Multi-agent ADNet.

procedure MATRACKER($s_{1,l}^{(i)}, s_{1,l}^{(-i)}, F_l$)
  $t \leftarrow 1$
  repeat
    Compute message $o_{t,l}^{(-i)}$ using $s_{t,l}^{(-i)}$
    $a_{t,l}^{(i)} \leftarrow arg\max_a p(a|s_{t,l}^{(i)}, o_{t,l}^{(i)}; \theta)$
    Set next state $s_{t+1,l}^{(i)}$ by state-transition function
    $t \leftarrow t + 1$
  until $s_{t,l}^{(i)}$ is a terminal state
  Set termination step $T_l \leftarrow t$
  Return $\{s_{t,l}^{(i)}\}, \{a_{t,l}^{(i)}\}$
end procedure

Chapter 5

Experiments

5.1 Action-Driven Visual Tracking

In this section, we validated the action-decision network (ADNet) proposed in Chapter 3. The ADNet aims to solve the problem of visual object tracking, that is, the tracking objective is a *single target tracking*. The experiments were conducted on the following specifications: i7-4790K CPU, 32 GB RAM, and GTX TITAN X GPU using MATLAB2016b and MatConvNet toolbox [61]. In our settings, ADNet and ADNet-fast run at 3 fps and 15 fps on the GPU, respectively.

In the followings, Section 5.1.1 describes the dataset used to evaluating the proposed method. In Section 5.1.2, we validated the effectiveness of ADNet by demonstrating various self-comparisons. In Section 5.1.3, we remarked the experimental results. Section 5.1.4, we evaluated the ADNet on the popular visual tracking benchmarks, Object Tracking Benchmark (OTB) [10, 62], comparing with existing trackers.

Table 5.1: Sequence Attributes of OTB Dataset. Descriptions are from the original paper [10].

Attributes	Description
Illumination Variation	The illumination in the target region is significantly changed
Scale Variation	The ratio of the bounding boxes of the first frame and the current frame is out of range.
Occlusion	The target is partially or fully occluded
Deformation	Non-rigid object deformation
Motion blur	The target region is blurred due to the motion of the target or the camera
Fast Motion	The motion of the ground truth is larger than t_m pixels ($t_m = 20$)
In-Plane Rotation	The target rotates in the image plane
Out-of-Plane Rotation	The target rotates out of the image plane
Out-of-View	Some portion of the target leaves the view
Background Clutters	The background near the target has similar color or texture as the target
Low Resolution	The number of pixels inside the ground-truth bounding box is less than $t_r(t_r = 400)$

5.1.1 Datasets

We evaluated our method on two OTB datasets: OTB-50 [62], which has 50 video sequences, and OTB-100 [10], which has 100 video sequences including OTB-50. The videos in the OTB datasets have various attributes, including illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters, and low resolution, which represent the challenging aspects in visual tracking. The detailed descriptions are presented in Table 5.1. In order to pre-train ADNet, we used 360 training videos from VOT2013 [63], VOT2014 [15], VOT2015 [22], and ALOV300 [21], in which videos overlapping with OTB-50 and OTB-100 were excluded.

The tracking performance was measured by conducting a one-pass evaluation (OPE) based on two metrics: center location error and overlap ratio [62]. The OPE is an evaluation way that initializes with the ground truth position in the first frame and runs a tracker throughout a test sequence. The center location error measures the distance between the center of the tracked frame and the ground truth and the bounding box overlap ratio measures the Intersection-over-Union (IOU) ratio between the tracked bounding box and the ground truth.

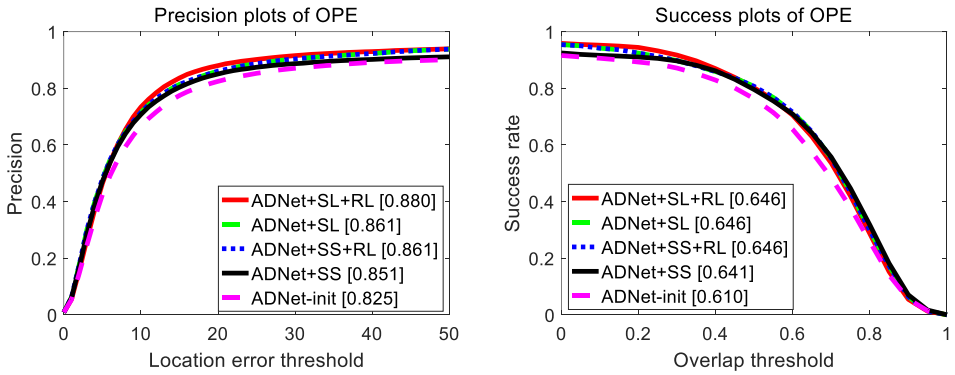


Figure 5.1: Self-comparison results of experiments on OTB-100. The scores in the legend indicate the mean precisions when the location error threshold is 20 pixel for the precision plots and area-under-curve (AUC) for the success plots.

5.1.2 Self-comparison

To verify the effectiveness of the components of ADNet, we conducted four variants of ADNet and evaluated them using OTB-100. We first conducted the baseline ‘’‘ADNet-init”, which is not pre-trained and simply uses the initial parameters. In ADNet-init, the parameters of convolutional networks (conv1-3) are initialized with the VGG-M [59] model, and the fully-connected layers (fc4-7) are initialized with random noises. “ADNet+SL” is the pre-trained models with supervised learning using fully labeled frames of the training sequences. “ADNet+SS” is trained using partially labeled data in the semi-supervised (SS) settings. In the training of ADNet+SS, the ground truth annotation is provided only every 10 frames. Then we conducted “ADNet+SL+RL” and “ADNet+SS+RL” by training ADNet+SL and ADNet+SS using reinforcement learning (RL), respectively. ADNet+SL+RL is the final version of the proposed method.

The precision and success rate of the self-comparisons are illustrated in Figure 5.1. By conducting SL, ADNet+SL and ADNet+SS achieved 3.6% and 2.6% improvement in precision performance improvement, respectively, compared to ADNet-init. In the semi-supervised case, the precision is 1.0% lower than that in the supervised case because of the lack of ground truth annotations. When conducting RL, ADNet+SL+RL and ADNet+SS+RL gained 1.9% and 1.0% additional improvement in precision performance to ADNet+SL and ADNet+SS, respectively. The results show that the RL can improve performance not only the semi-supervised case but also the supervised case.

5.1.3 Analysis

5.1.3.1 Movement Step Size

Since the action of our decision process is defined in discrete space, the tracker moves with a discrete amount of step size. The movement step size, which determines how

Table 5.2: Experimental results with three movement sizes (2%, 3%, and 4%) on OTB-100 dataset.

Movement size	Precision (20px)	Speed (FPS)
2%	86.5	2.83
3%	88.0	2.90
4%	86.8	2.96

much to move by an action, has a trade-off between computational complexity and tracking accuracy. If the size is too small, the number of iterations to capture the target increases and the tracker may not cope with fast motion of the target. In contrast, if it is too large, the accuracy is not precise when the tracking action is terminated by the stop action. In this work, the movement size was determined in proportion to the object size to control the number of iterations regardless of the size of the target and was empirically set to 3% of the width and height of the current target.

Table 5.2 shows the experiments on the OTB-100 dataset to examine the tracking precision when changing the movement step size to 2%, 3%, and 4% of the width and height. As shown in Table 5.2, tracking speed gets higher when the step size is increasing and the tracking accuracy gets degradations when the step size is smaller (2%) or larger (4%) than the selected step size 3%.

5.1.3.2 Action Dynamics Vector

The goal of the action dynamics vector is to consider the movement dynamics of the target object by utilizing the past actions when deciding the current action. As described in Section 3.2, the action dynamics vector is encoded by concatenating past 10 actions $d_{t,l} \in \mathbb{R}^{110}$ and the network output has 11-dimensional action probability. Therefore, the network weights connected from action dynamics to output nodes in

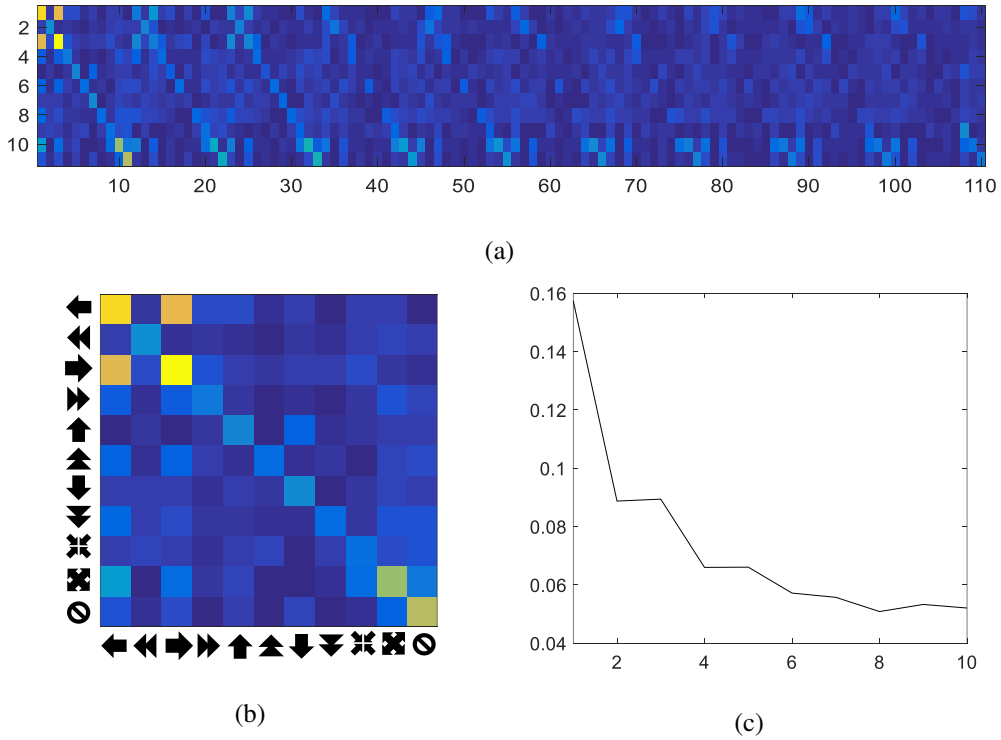


Figure 5.2: Analysis of the network weights of action dynamics vector to decide the current action. **(a)** The squared weight values from action dynamics (110-dim) to action outputs (11-dim). **(b)** The first action block of the weights. **(c)** The sum of squares of the weight values for action dynamics.

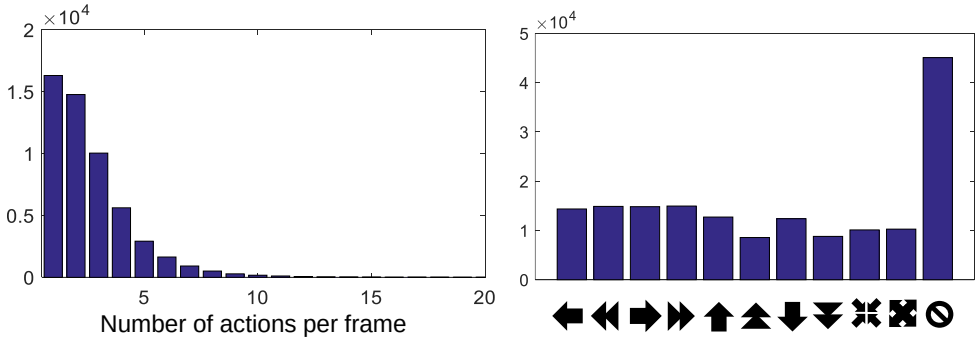
fc6 layer can be represented as a matrix $W_{dyn} \in \mathbb{R}^{110\times11}$. To investigate the effect of the action dynamics feature on the current action, we plotted the squared values of W_{dyn} in Figure 5.2 (a) where higher values are indicated by brighter colors and lower values by darker ones. The x-axis denotes the index of the action dynamics vector and the y-axis denotes the index of the output action node. In Figure 5.2 (a), the 11 by 11 blocks corresponding to 10 actions in the past are shown, where each block indicates the effect of the past action on the action output.

For more detailed analysis, we plotted the first block of W_{dyn}, corresponding to the action just before the current state in Figure 5.2 (b). The x-axis and y-axis respectively represent the past and current action. For example, the first row denotes the connected weights from the past action to select the 'left' action. The higher diagonal elements indicates that the existence of the same action in the past plays an important role to determine the current action. From the first row, the weight connected to the previous 'right' action also has a high value to select the 'left' action in the current step. In other word, to select an action, the selection of the opposite action in the previous state also has an important role.

In addition, we measure the influence of the past action by summing the elements in each block of W_{dyn} as follows,

$$\text{Influence of the past } k^{th} \text{ action} = \sum_{w \in W_{dyn}[k]} w^2, \tag{5.1}$$

where $W_{dyn}[k]$ denotes the k^{th} block of the matrix W_{dyn}. The influence of the past action is plotted in Figure 5.2 (c). The influence value is decreasing when the element index of the action dynamics is increasing in x-axis, that is, the more past action has the weaker influence to decide the current action.



(a) (b)

Figure 5.3: Histograms of the number of moves per frame in **(a)** and the number of selections for each action in **(b)** in the test sequences.

5.1.3.3 Efficiency of the Searching Strategy

Figure 5.3 shows the histograms of the number of moves per frame and the number of selections for each action in the test sequences. As shown in Figure 5.3 (a), the ratio of the frames requiring more than five actions to capture the target to the whole frames was only around 7%. That is, most of the frames require fewer than five actions to pursue the target in each frame. The proposed tracker occasionally conduct re-detection, and the ratio of the frames using re-detection to the whole frames is around 9%. The average number of searching steps including the required actions and the candidates by re-detection is 28.26 per frame, which is much smaller than that of state-of-the-art trackers such as MDNet [13]($= 256$ per frame).

We also plot how many times each action was selected during the test sequences in Figure 5.3 (b). With the movement size and action types defined by the proposed method, the numbers of selected actions are uniformly distributed except stop action. The number of selection of stop action is much higher than others since stop action is always selected at each frame.

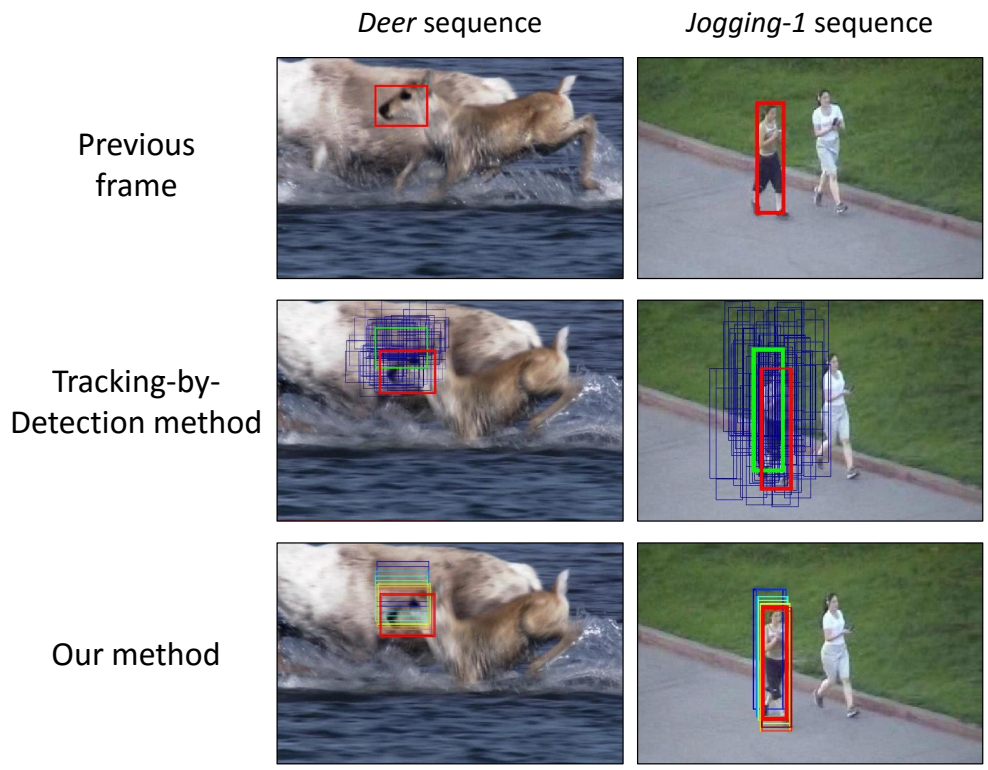
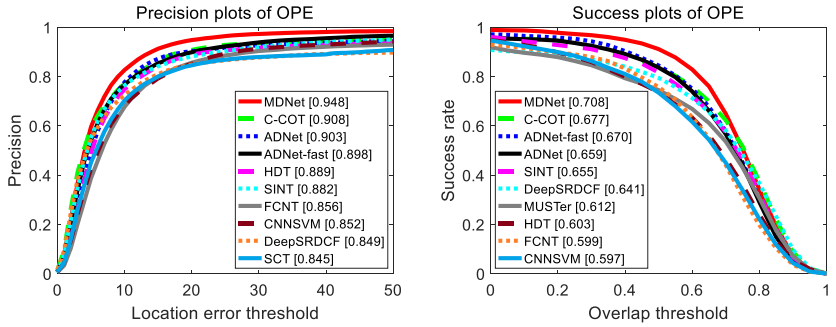


Figure 5.4: Searching strategy comparison between the existing tracking-by-detection approach [13] (second column) and the proposed method (third column) on the *Deer* and *Jogging-1* sequences.

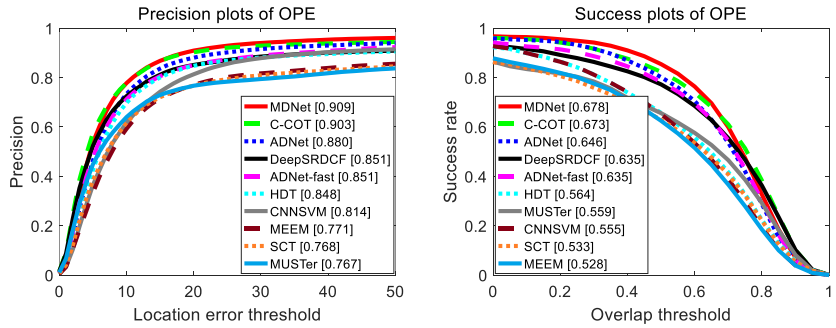
Table 5.3: Summary of experiments on OTB-100.

	Algorithm	Prec.(20px)	IOU(AUC)	FPS	GPU
	ADNet	88.0%	0.646	2.9	O
	ADNet-fast	85.1%	0.635	15.0	O
Non real-time	MDNet [13]	90.9%	0.678	< 1	O
	C-COT [25]	90.3%	0.673	< 1	O
	DeepSRDCF [24]	85.1%	0.635	< 1	O
	HDT [64]	84.8%	0.564	5.8	O
	MUSTer [8]	76.7%	0.528	3.9	X
Real-time	MEEM [65]	77.1%	0.528	19.5	X
	SCT [9]	76.8%	0.533	40.0	X
	KCF [7]	69.7%	0.479	223	X
	DSST [6]	69.3%	0.520	25.4	X
	GOTURN [31]	56.5%	0.425	125	O

Figure 5.4 qualitatively illustrates the efficiency of the proposed method pursuing the target by sequential actions, compared to the existing tracker based on tracking-by-detection strategy [13]. The second row plots the results of tracking-by-detection method and the green, red, and blue boxes denote the previous target position, the current target position, and the generated target candidates respectively. The third row shows the tracking process of the proposed method by selecting sequential actions.



(a) OTB-50



(b) OTB-100

Figure 5.5: Precision and success plots on OTB-50 [62] and OTB-100 [10]. Only the top 10 trackers are presented.

5.1.4 State-of-the-art Comparison

We compared ADNet in a comprehensive comparison with 13 state-of-the-art trackers including MDNet [13], C-COT [25], GOTURN [31], HDT [64], DeepSRDCF [24], SINT [30], FCNT [29], SCT [9], MUSTer [8], CNN-SVM [12], MEEM [65], DSST [6], and KCF [7]. We evaluated the GOTURN tracker [31] on the OTB dataset using the author's code.

Figure 5.5 shows the plots of precision and success rate based on center location error and overlap ratio respectively and Table 5.3 summarizes the comparison of tracking performance with computational speed (fps) and GPU usage. The proposed method shows comparable performance with the state-of-the-art trackers, MDNet [13] and C-COT [25] in both precision and success rate. The proposed method is much efficient in computation, where ADNet is about three times faster than MDNet and C-COT. ADNet-fast, the fast version of ADNet, has a 3% performance degradation but runs in real-time (15 fps) and shows performance comparable with those of other CNN-based trackers such as DeepSRDCF [24] and HDT [64]. As shown in Table. 5.3, ADNet-fast achieved the best performance among the real-time tracking algorithms.

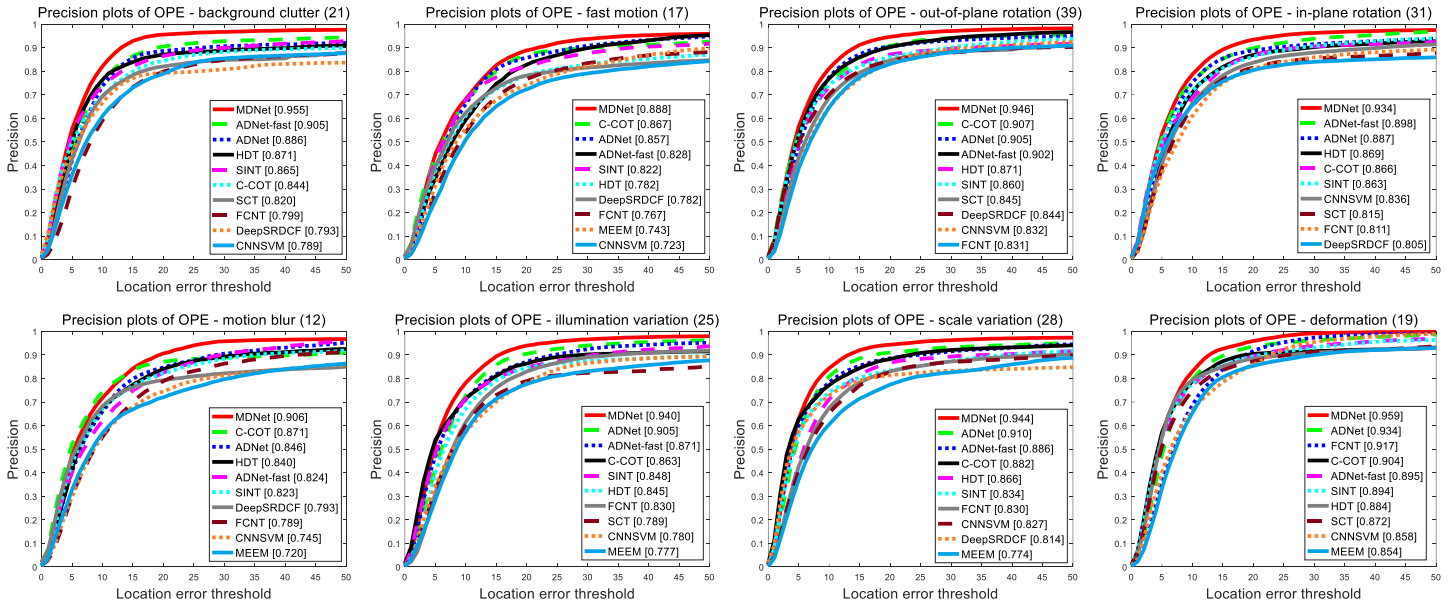


Figure 5.6: Precision plots on OTB-50 for the subset of challenging attributes: *background clutter, fast motion, out-of-plane rotation, in-plane rotation, motion blur, illumination variation, scale variance, and deformation*. The plotted curves are based on the center location error. Only the top 10 trackers are presented.

Figure 5.6 shows the tracking performance on various challenging tracking situations such as background clutter, fast motion, deformation, etc. The proposed methods, ADNet and ADNet-fast, demonstrate robust tracking performance to handle the challenging situations where high-level semantics and contextual information are required.

5.1.5 Qualitative Results

The qualitative results of the proposed method are illustrated in Figure 5.7. The results compare the proposed method (ADNet-fast) with the real-time state-of-the-art trackers, GOTURN [31], MEEM [65], KCF [7], and SCT [9]. The results show the robustness of the trackers in the challenging sequences where the scale variation, illumination change, occlusion, and deformation are appeared. We plotted critical frames in the test sequences where the proposed tracker successfully capture the target while the other trackers often fail to track. The existing real-time state-of-the-art trackers have difficulty to track in the complex scene as in the Figure 5.7 since they use lightweight features or simple tracking models to obtain fast speed. In contrast, the proposed method can achieves satisfactory tracking performance as well as real-time speed since an efficient action-driven search strategy is adopted even for relatively complicated deep networks.

Figure 5.8 presents a few failure cases of the proposed method. ADnet failed to follow the abrupt movement of the target in *Ironman* sequence, and the proposed actions could not adapt to the dramatic aspect ratio change in *Diving* sequence.

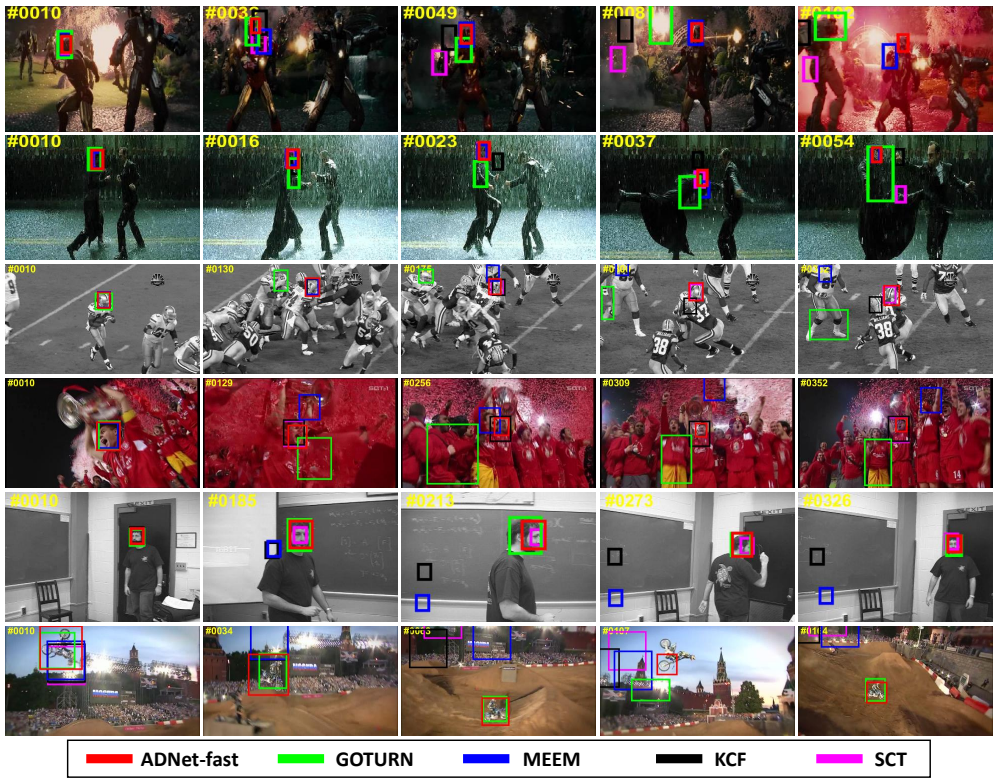


Figure 5.7: Qualitative results of the proposed method. The test sequences are *Ironman, Matrix, Football, Soccer, Freeman1,* and *MotorRolling*.

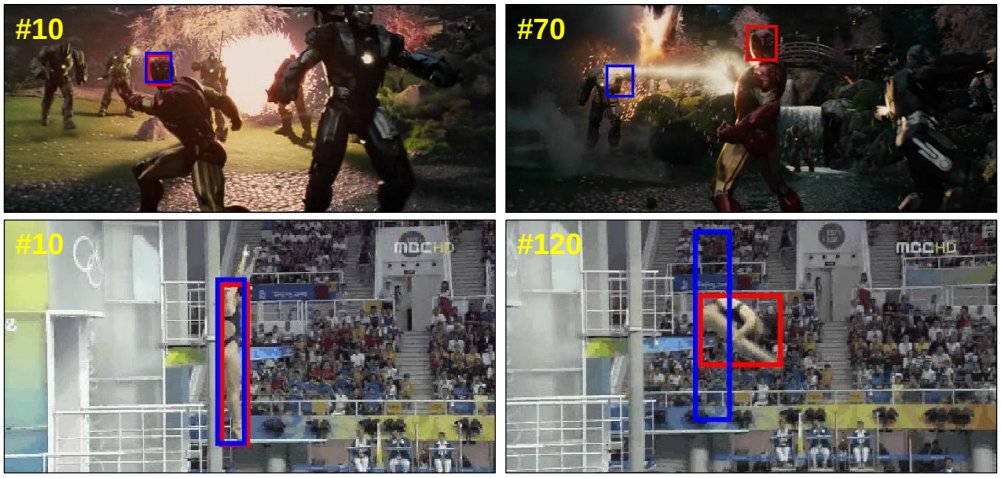


Figure 5.8: Failure cases of the proposed method on *Ironman* and *Diving* sequences. Blue and red bounding boxes indicate the ground truths and the tracking results of ADNet, respectively.

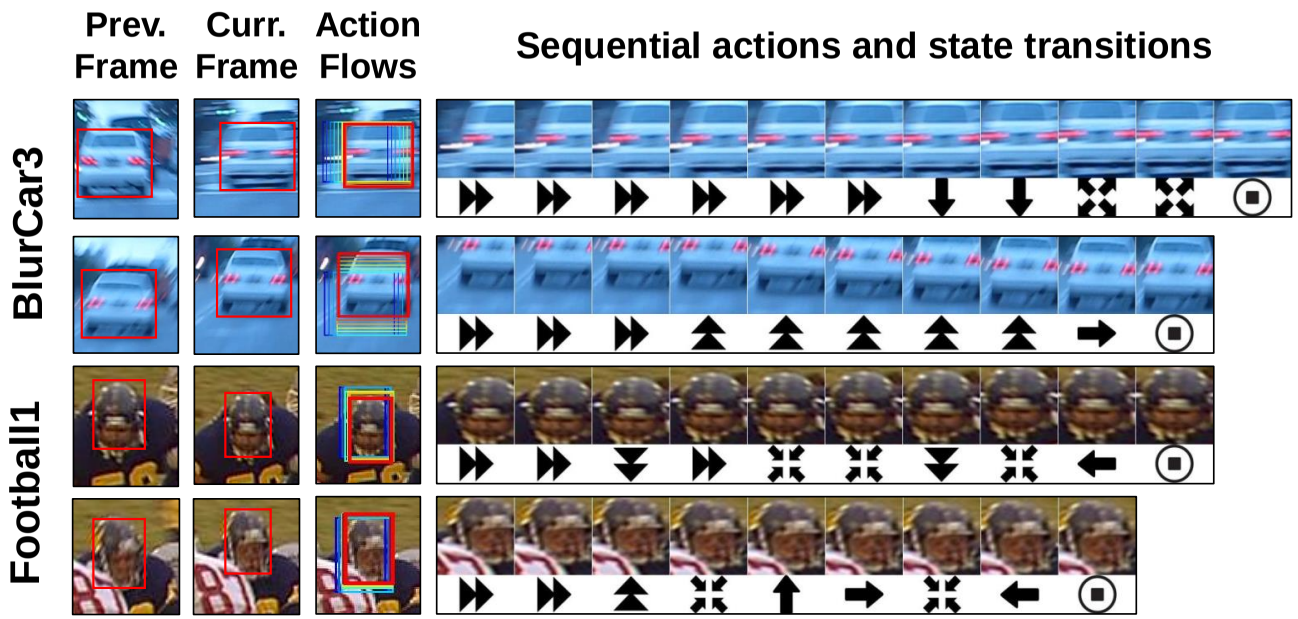


Figure 5.9: Examples of sequential actions selected by ADNet on *BlurCar3*, and *Football1* sequences. The action flows from the previous position to the current position and the state transitions are presented.

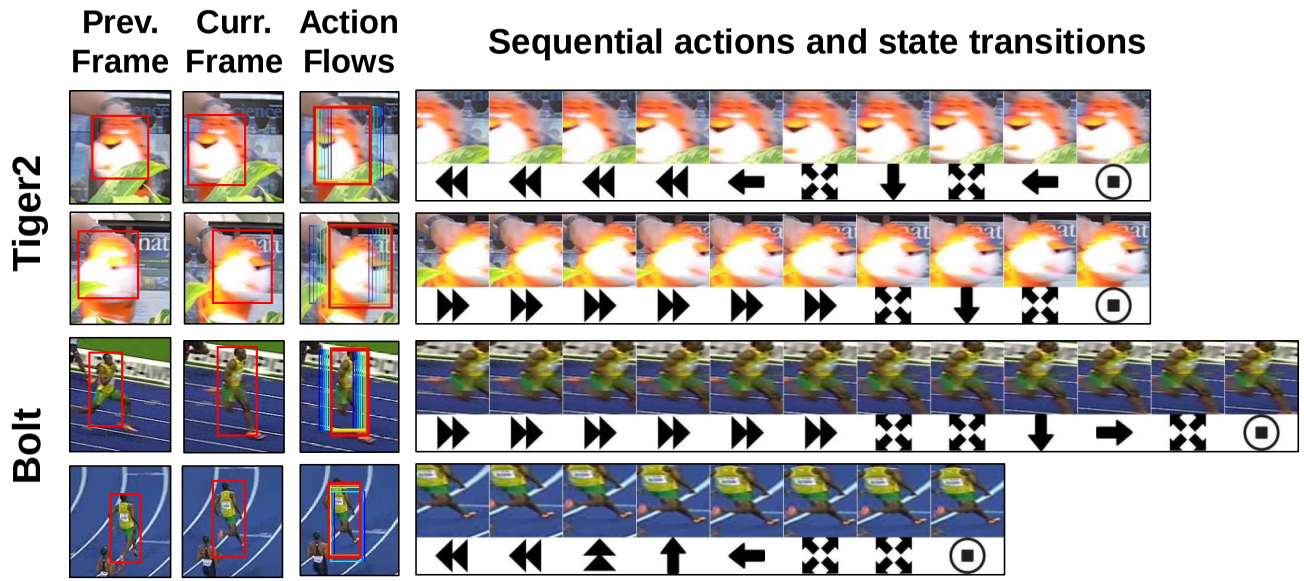


Figure 5.10: Examples of sequential actions selected by ADNet on *Tiger2*, and *Bolt* sequences. The action flows from the previous position to the current position and the state transitions are presented.

Figure 5.9 and Figure 5.10 illustrate examples of the sequential actions selected by the proposed method. The first column shows the target position in the previous frame and the second column shows the current position of the target. The bounding box flows from the previous position (blue) to the captured target position (red) are shown in the third column. In the remaining part of Figures 5.9 and 5.10, the sequential transitions of the state and the corresponding actions are represented by the image patches and the action symbols. In the sequential state transition, we can see that the agent selects proper actions to capture the target in each state.

5.2 Interacted Action-Driven Tracking

In this section, we evaluated the multi-agent ADNet for interacted action-driven tracking proposed in Chapter 4. The proposed multi-agent ADNet aims to track the target objects in a complex scene with considering of the movements of other objects. We compared the multi-agent ADNet with the baseline, adopting multiple independent-ADNet and the state-of-the-art tracking method.

5.2.1 Datasets

We evaluated our method on APIDIS dataset [66] which has 7 videos sequences with total 10,500 frames of basketball scene including various interacting movements among many players. All the objects such as balls, referees, and players in the frames are fully annotated by human. As shown in Figure 5.11, the videos in the APIDIS dataset are challenging for visual tracking task due to severe occlusions and deformations. The videos captured the scene playing basketball using 7 different camera views. To avoid overfitting problem, training set and the test set were fully disjointed, where we selected 441 frames of *cam6* as the training set and 615 frames of *cam1* as the test set. The selected 1056 frames have about 10 basketball players moving with mutual interactions.

To evaluate the proposed method, we split the test frames as 30 sequences whose length are 100 frames with temporal overlapping. Therefore, the proposed method is evaluated using 30 test sequences where the number of target objects are about 10 at each sequence. The tracking performance was measured by conducting a one-pass evaluation (OPE) based on two metrics: center location error and overlap ratio [62]. The OPE is an evaluation way that initializes the tracking position with the ground truth in the first frame and runs a tracker throughout a test sequence. The center location error measures the distance between the center of the tracked frame and the

Figure 5.11: APIDIS dataset example frames. Left: #345 frame of *cam6*, Right: #843 frame of *cam1*.

ground truth and the bounding box overlap ratio measures the Intersection-over-Union (IOU) ratio between the tracked bounding box and the ground truth.

Table 5.4: Experiments with various ROI ratio.

ROI ratio	Precision (20px)	IoU (AUC)
0.5	73.2	48.7
1.5	**75.0**	50.0
2.0	74.5	**50.1**
2.5	73.9	49.4

Table 5.5: Summary of experiments on APIDIS dataset.

Algorithm	Precision (20px)	IoU (AUC)	Proc. time (sec/frame)
MAADNet	**75.0**	**51.0**	2.5
i-ADNet	68.8	46.1	2.3
i-MDNet [13]	68.9	46.5	8.3

5.2.2 Self Comparison

We investigate the effect of ROI ratio in the message encoding module. Table 5.4 shows the results of precision and IOU when changing the ROI ratio from 0.5 to 2.5. The size of ROI can be obtained by multiplying the size of the target position with the ROI ratio.

As shown in Table 5.4, increasing ROI ratio can improve the tracking performance. However, when the ROI ratio is over 2.0, the performance degrades. Since the message encoder aligns the messages by 3×3 grid, it is difficult to understand the state of surrounding objects by setting too wide ROI area. Therefore, we empirically choose the ROI ratio 1.5 for the experiments.

5.2.3 Quantitative Results

In this section, we compared the multi-agent ADNet with the single-agent ADNet and MDNet [13] which achieved the state-of-the-art tracking performance in a single target tracking. Since ADNet and MDNet [13] aim to track a single object target, the methods are performed in a parallel manner on the test sequences. We referred them as 'isolated-ADNet' and 'isolated-MDNet'.

Figure 5.12 and Table 5.5 show the results of average precision and success rate for multiple targets based on center location error and overlap ratio respectively. The isolated trackers, i-ADNet and i-MDNet, are not much different in performance. The proposed method, multi-agent ADNet (MA-ADNet), achieved better performance than the isolated state-of-the-art trackers, with improvement about 6%. The MAADNet has inter-agent communications which plays an important role when objects are interfered by each other as shown in APIDIS dataset. The processing time of the interacted tracker, i-ADNet, and i-MDNet are 2.3, 2.5, and 8.3 seconds per frame, respectively. We can see that the multi-agent ADNet has little loss of overall tracking speed even

considering the interaction among agents. That is, the message encoder and the action selector modules do not have a significant impact on overall speed and rather promote the tracking actions with the help of neighboring agents.

5.2.4 Qualitative Results

Figure 5.13 shows the examples of selected actions by considering the interactions among agents. The current target positions are indicated by green boxes and the ground truths are represented as red boxes. We plotted the action from the target agent, which is denoted by 'T-Action', and the action from the neighboring agents, which is denoted by 'N-Action'. By the action selector of the proposed MA-ADNet tracker, the finally selected actions is indicated by blue boxes. In Figure 5.13, the examples of the first row show that the action selector successfully decides the right actions. We also plotted some failure cases of the action selector on the second row in Figure 5.13 where the action selector was confused and made wrong selection.

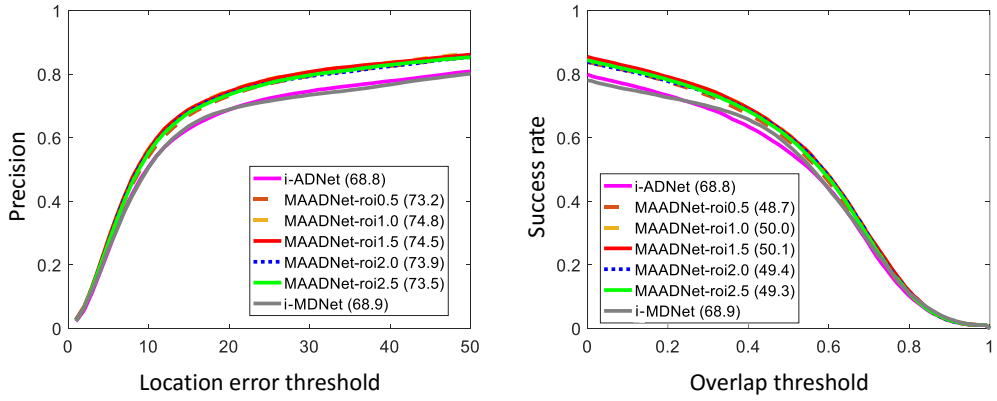


Figure 5.12: Precision and success plots on APIDIS dataset. We plotted the experimental results of the ablation tests and the isolated state-of-the-art methods.

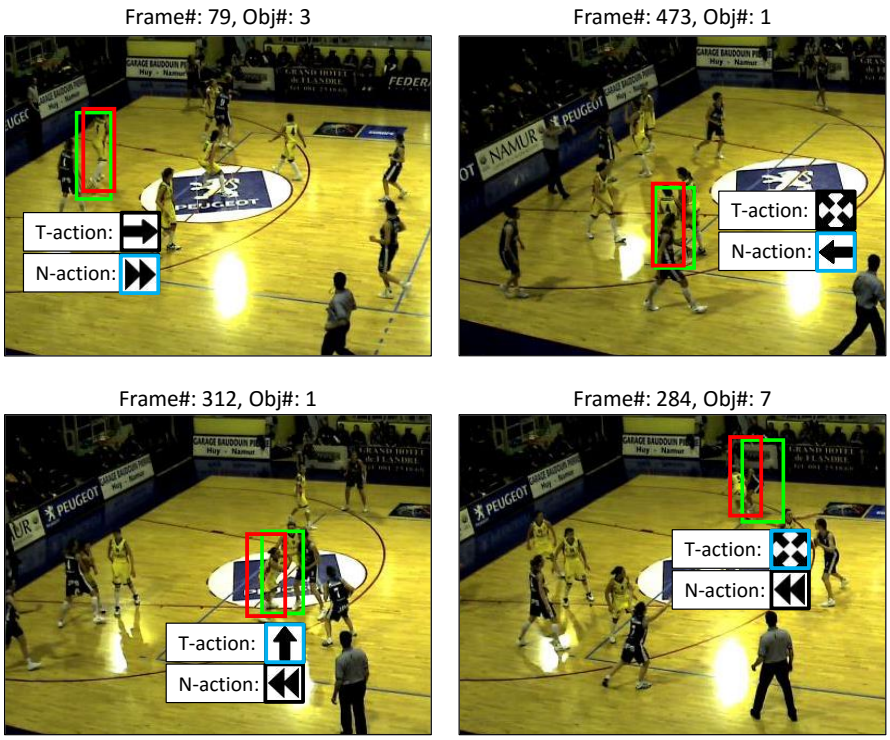


Figure 5.13: Examples of selected actions by considering the interactions among agents. The green boxes indicate the current target position and the red boxes the ground truth target positions. We plotted the two actions, 'T-Action' and 'N-Action', which represent the action from target agent and the action from other agents, respectively. The action selected by the action selector module is indicated by blue boxes.

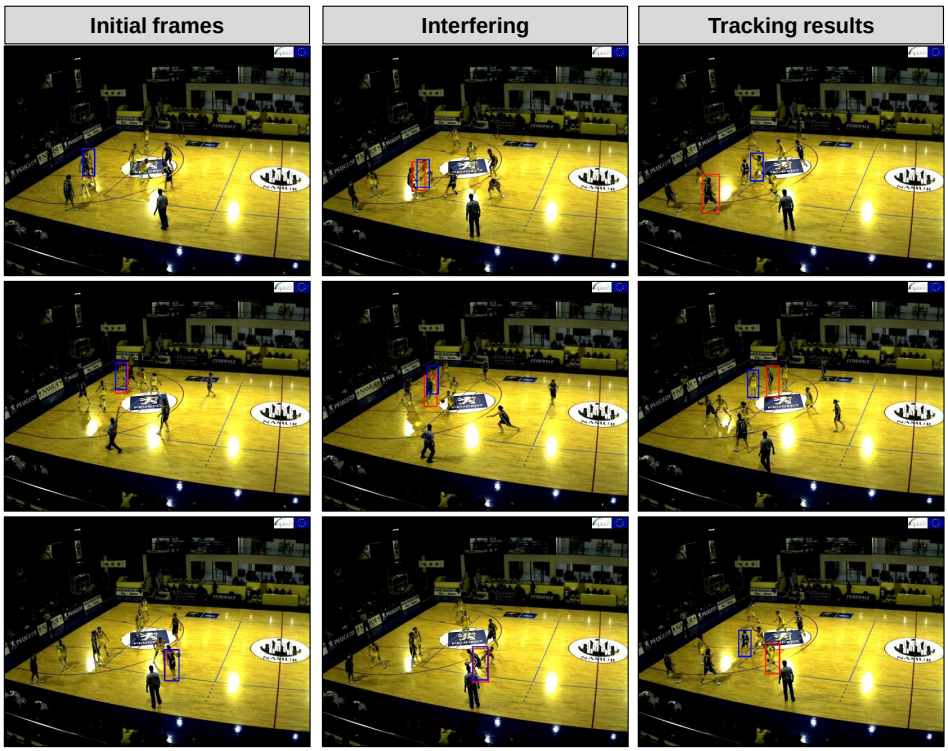


Figure 5.14: Qualitative results on APIDIS dataset. The blue and red bounding boxes represent the position of tracker, indep-ADNet and multi-agent ADNet, respectively.

Figure 5.14 shows the qualitative results of the proposed method. The tracking results of the proposed method are denoted by blue bounding boxes and the red bounding boxes represents the results of the isolated ADNet. The first column shows the initial frames and the second column highlights the moment of interference by neighborhood objects. The last column shows the tracking results after the interference and the results shows that the proposed method maintains the tracking targets while the isolated ADNet failed to track the target.

Chapter 6

Conclusion

6.1 Concluding Remarks

In this dissertation, we have proposed a novel action-driven method using deep convolutional networks for visual tracking. The proposed tracker is controlled by action-decision network (ADNet), which pursues the target object by sequential actions iteratively. To the best of our knowledge, it is the first attempt to adopt the action-driven tracking strategy controlled by pursuing actions trained by deep reinforcement learning. Action-driven tracking makes a significant contribution to the reduction of computation complexity in tracking. In addition, the reinforcement learning makes it possible to use the partially labeled data, which could greatly contribute to building of training data with little effort. According to the evaluation results, the proposed tracker achieves a state-of-the-art performance in 3 fps, which is three times faster than the traditional deep network-based trackers employing a tracking-by-detection strategy. Furthermore, the fast version of the proposed tracker achieves a real-time speed (15 fps), by adjusting meta-parameters of the ADNet, with accuracy that outperforms state-of-the-art

real-time trackers.

We also extend the action-driven tracker to a multi-agent architecture composed of interacted multiple trackers, which aims to deal with the interaction among multiple objects. Each tracker in the multi-agent architecture determines an action to capture the target by considering the target object and its neighboring objects at the same time. The multi-agent architecture is designed to learn the policy to determine joint actions of multiple agents utilizing deep reinforcement learning. To encode the contextual information of neighboring objects and determine appropriate actions, we newly proposed the message encoder and the action selector. This allows performing interacted action for tracking with the advantage of the action-driven tracking method. The experimental results show that the proposed method achieved the better performance than conventional trackers in complex scenes where several objects are interfering.

6.2 Future Works

The action-driven approach proposed in this dissertation is faster and more efficient than other deep network-based tracking methods. The proposed tracker achieved the real-time performance in the single agent case, but in the multi-agent case, the real-time performance is not achieved. In real-world applications, such as autonomous driving, visual surveillance, or analysis of crowd people, multiple target tracking is a critical problem. Moreover, since it is hard to know the information about the objects appearing and disappearing from the sequences, the help of the object detector is needed to track robustly in situations where the number of objects is changing. In recent years, fast and accurate object detectors, such as SSD [67] or YOLO detector [68], have been developed and they enable real-time multiple object detection. Therefore, for the future work, it is interesting to work for real-time multiple object tracking scheme which maintains the effectiveness of action-driven approach with the help of real-time multi-

object detection methods. In addition, our final goal is to develop a new deep learning architecture for the synergistic framework of detection and tracking schemes.

Bibliography

[1] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting." in *BMVC*, vol. 1, no. 5, 2006, p. 6.

[2] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised on-line boosting for robust tracking," in *European conference on computer vision*. Springer, 2008, pp. 234–247.

[3] B. Babenko, M.-H. Yang, and S. Belongie, "Robust object tracking with online multiple instance learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1619–1632, 2011.

[4] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[5] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2544–2550.

[6] M. Danelljan, G. Häger, F. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," in *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.

[7] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[8] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, "Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 749–758.

[9] J. Choi, H. Jin Chang, J. Jeong, Y. Demiris, and J. Young Choi, "Visual tracking using attention-modulated disintegration and integration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4321–4330.

[10] Y. Wu, J. Lim, and M.-H. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, 2015.

[11] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung, "Transferring rich feature hierarchies for robust visual tracking," *arXiv preprint arXiv:1501.04587*, 2015.

[12] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," *arXiv preprint arXiv:1502.06796*, 2015.

[13] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," *arXiv preprint arXiv:1510.07945*, 2015.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, and Ma, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[15] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Cehovin, G. Nebehay, T. Vojir, and G. Fernandez, "The visual object tracking VOT2014 challenge results," in *Workshop on the VOT2014 Visual Object Tracking Challenge*, Sep. 2014, pp. 191–217.

[16] S. Li, O. Wu, C. Zhu, and H. Chang, "Visual object tracking using spatial context information and global tracking skills," *Computer Vision and Image Understanding*, vol. 125, pp. 1–15, 2014.

[17] S. Duffner and C. Garcia, "Exploiting contextual motion cues for visual object tracking," in *European Conference on Computer Vision*. Springer, 2014, pp. 232–243.

[18] X. Wang, E. Türetken, F. Fleuret, and P. Fua, "Tracking interacting objects optimally using integer programming," in *European Conference on Computer Vision*. Springer, 2014, pp. 17–32.

[19] ——, "Tracking interacting objects using intertwined flows," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 11, pp. 2312–2326, 2016.

[20] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, vol. 74, no. 18, pp. 3823–3831, 2011.

[21] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1442–1468, 2014.

[22] M. Kristan, J. Matas, A. Leonardis, and Felsberg, "The visual object tracking vot2015 challenge results," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 1–23.

[23] J. Choi, H. Jin Chang, S. Yun, T. Fischer, Y. Demiris, and J. Young Choi, "Attentional correlation filter network for adaptive visual tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[24] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 58–66.

[25] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *ECCV*, 2016.

[26] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," in *Advances in neural information processing systems*, 2013, pp. 809–817.

[27] H. Li, Y. Li, and F. Porikli, "Robust online visual tracking with a single convolutional neural network," in *Asian Conference on Computer Vision*. Springer, 2014, pp. 194–209.

[28] ——, "Deeptrack: Learning discriminative feature representations online for robust visual tracking," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1834–1848, 2016.

[29] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3119–3127.

[30] R. Tao, E. Gavves, and A. W. Smeulders, "Siamese instance search for tracking," *arXiv preprint arXiv:1605.05863*, 2016.

[31] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," *arXiv preprint arXiv:1604.01802*, 2016.

[32] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[33] A. Gonzalez-Garcia, A. Vezhnevets, and V. Ferrari, "An active search strategy for efficient object class detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3022–3031.

[34] D. Yoo, S. Park, J.-Y. Lee, A. S. Paek, and I. So Kweon, "Attentionnet: Aggregating weak directions for accurate object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2659–2667.

[35] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2488–2496.

[36] S. Mathe, A. Pirinen, and C. Sminchisescu, "Reinforcement learning for visual object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2894–2902.

[37] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4705–4713.

[38] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[39] R. S. Sutton, *Introduction to reinforcement learning*, vol. 135.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[41] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR, abs/1509.06461*, 2015.

[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[43] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[44] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[45] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *arXiv preprint arXiv:1603.00748*, 2016.

[46] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[47] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[48] D. Jayaraman and K. Grauman, "Look-ahead before you leap: end-to-end active recognition by forecasting the effect of motion," *arXiv preprint arXiv:1605.00164*, 2016.

[49] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[50] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[51] C. Zhang and V. Lesser, "Coordinating multi-agent reinforcement learning with limited communication," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1101–1108.

[52] T. Kasai, H. Tenmoto, and A. Kamiya, "Learning of communication codes in multi-agent reinforcement learning problem," in *Soft Computing in Industrial Applications, 2008. SMCia'08. IEEE Conference on*. IEEE, 2008, pp. 1–6.

[53] J. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[54] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé III, "Opponent modeling in deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 1804–1813.

[55] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[56] J. Foerster, N. Nardelli, G. Farquhar, P. Torr, P. Kohli, S. Whiteson *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1702.08887*, 2017.

[57] H. M. Le, P. Carr, Y. Yue, and P. Lucey, "Data-driven ghosting using deep imitation learning," 2017.

[58] X. Kong, B. Xin, Y. Wang, and G. Hua, "Collaborative deep reinforcement learning for joint object search," *arXiv preprint arXiv:1702.05573*, 2017.

[59] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *arXiv preprint arXiv:1405.3531*, 2014.

[60] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1471–1530, 2004.

[61] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[62] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2411–2418.

[63] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Cehovin, G. Nebehay, G. Fernandez, T. Vojir, A. Gatt *et al.*, "The visual object tracking vot2013 challenge results," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2013, pp. 98–111.

[64] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, and J. L. M.-H. Yang, "Hedged deep tracking," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[65] J. Zhang, S. Ma, and S. Sclaroff, "Meem: robust tracking via multiple experts using entropy minimization," in *European Conference on Computer Vision*. Springer, 2014, pp. 188–203.

[66] C. De Vleeschouwer, F. Chen, D. Delannay, C. Parisot, C. Chaudy, E. Martrou, A. Cavallaro *et al.*, "Distributed video acquisition and annotation for sport-event summarization," *NEM summit*, vol. 8, 2008.

[67] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[68] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

Abstract

본 학위 논문은 심층 강화 학습에 의해 학습된 행동을 순차적으로 선택하여 영상의 객체를 추적하는 새로운 추적 알고리즘을 제안한다. 최근의 심층 신경망을 이용한 추적 방법들은 추적 대상의 위치를 찾기 위해 여러 후보 위치 중 가장 일치하는 점수를 가진 후보를 선택하는 탐지에 의한 추적(tracking-by-detection) 방법을 채택한다. 탐지에 의한 추적은 간단한 방식으로 좋은 성능을 보이지만, 후보 위치를 비효율적으로 탐색하는 문제가 있다. 우리는 관심 물체를 지속적으로 주시하는 연속적인 행동에 의해 대상을 추적하도록 제어하는 행동 중심의 추적 방법을 제안한다. 제안하는 방법은 심층 강화 학습에 의해 학습되며, 기존의 심층 신경망을 이용한 추적기들에 비해 적은 연산량을 가지며 우수한 추적 성능을 보인다. 행동을 제어하기 위한 심층 신경망은 여러 학습 동영상을 이용하여 사전 학습(pre-train)되며, 추적 상황에서 대상 물체 및 배경의 변화에 적응하기 위해 온라인으로 미세학습(fine-tune)된다. 제안하는 사전 학습 방법은 감독 학습과 강화 학습이 결합된 형태로, 부분적으로 레이블된 데이터들도 준 감독 학습(semi-supervised learning)에 성공적으로 활용될 수 있다. 또한, 본 논문은 제안된 행동 중심의 추적 방법을 다중 에이전트 시스템으로 확장하여 복잡한 상황에서 상호 작용하는 객체를 추적하는 문제를 해결한다. 제안된 다중 에이전트 추적 시스템의 구조는 복수개의 에이전트로부터 추적을 위한 적합한 행동을 동시에 선택하도록 설계하고 강화 학습을 통해 학습한다. 다양한 실험을 통해 제안된 추적 방법은 최신 심층 신경망 기반 추적기보다 빠른 속도와 우수한 추적 성능을 보이는 것을 검증하였다.

Keywords: 객체 추적, 합성곱 신경망, 심층 강화 학습

Student Number: 2013-30245